DIPLOMA IN INFORMATION COMMUNICATION TECHNOLOGY

SYSTEM ANAYSIS AND DESIGN (SAD)

TOPICS TO BE COVERED

Topic 1: Introduction to Systems Analysis and Design

Topic 2: Systems Theory/Concept

Topic 3: Systems Development Life Cycle (SDLC)

Topic 4: Problem Definition

Topic 5: Feasibility Study

Topic 6: Systems Analysis

Topic 7: Systems Design and Development

Topic 8: System Implementation

Topic 9: System Maintenance and Review

Topic 10: System Documentation

Topic 11: System Acquisition

Topic 11: ICT Project Management

Topic 12: Emerging Trends in SAD

TOPIC 1: INTRODUCTION TO SYSTEMS ANALYSIS AND DESIGN

THEORY

16.2.2.1. T0 Specific Objectives

By the end of this topic, the trainee should be able to:

a) explain meaning of SAD

b) describe the components of an information system

c) describe the roles of information systems stake holders

d) describe the types of information systems

CONTENT

Meaning of terms:

- **System**

A system is a set of an orderly grouping of interdependent components linked/put together according to a plan to achieve a specific objective. Examples include, computer system, music system etc.

- **Information**

This is the processed data that is meaningful to the user. It can be stored, as well as can be communicated

- **information system**

Is a collection of people, procedures, a base of data and (sometimes) hardware and software that collects, process, stores and communicates data for transaction processing at operational level and information to support management decision making.

- **information technology**

This is a term that encompasses all forms of technology used to create, store, exchange and utilize information in its various forms ie.

- Business data
- Motion pictures
- Multimedia presentations etc

Data:

These are the raw facts which do not have any form of meaning interms of alphanumeric (letter, and numbers) ,text and figures, special characters etc to the user.

Analysis:

This is the detailed assessment/study of the components of the existing systems and its requirements.

System Analysis:

Involves evaluation of the current system components and its requirements using gathered facts or information. One needs to evaluate whether the current and projected user needs are being met, if not he/she should give a recommendation of what ought to be done.

Objectives of systems analysis:

- To determine information needs of an organization and the users of the information
- Determination of the current activities of the system i.e functions involved in conversion of inputsto outputs
- Determination of the intended systems outputs
- Determination of the resources required for the intended system
- Determine capabilities required in the system to meet information needs of the organization.

Components of an information system

The components of an information includes:

- **People:** use the systems to fulfil their informational need. They include end users and operations personnel such as computer operators, system analyst, programmers, data administrators etc.
- **Computer hardware:**these are the physical computer equipment and devices, which provide for five major functions namely:
    - Input/or data entry
    - Output
    - Central processor (computational and control functions)
    - Secondary storage for data and programs
    - Communication
- **Computer software:** refers to the instructions that direct the operations of the computer hardware. Computer software is classified into systems software and application software.
    - System software examples include: OS, windows, linux, macintosh etc.
    - Examples of application software are the general purpose software (Microsoft office)
- **Database:** contains all data utilized by application software. An individual set of stored data is referred to as a file.
- **Procedure:** these are defined steps followed to carry out a given task. Procedures exist in physical forms as manuals or instruction booklets. Three major types of procedures are required.
    - User instructions for application users to record data, to use a terminal for data entry or retrieval
    - Instructions for preparation of input by data preparation personnel.
    - Operating instructions for computer operations personnel.

Types of information systems

- **Transaction processing systems**

What is a Transaction Processing System?

Transaction Processing System are operational-level systems at the bottom of the pyramid. They are usually operated directly by shop floor workers or front line staff, which provide the key data required to support the management of operations. This data is usually obtained through the automated or semi-automated tracking of low-level activities and basic transactions.

Functions of a TPS

TPS are ultimately little more than simple data processing systems.

Functions of a TPS in terms of data processing requirements		
Inputs	Processing	Outputs
Transactions Events	Validation Sorting Listing Merging Updating Calculation	Lists Detail reports Action reports Summary reports?

Some examples of TPS

- Payroll systems
- Order processing systems
- Reservation systems
- Stock control systems
- Systems for payments and funds transfers

The role of TPS

- Produce information for other systems
- Cross boundaries (internal and external)
- Used by operational personnel + supervisory levels
- Efficiency oriented

- **Management information systems**

What is a Management Information System?

For historical reasons, many of the different types of Information Systems found in commercial organizations are referred to as "Management Information Systems". However, within our pyramid model, Management Information Systems are management-level systems that are used by middle managers to help ensure the smooth running of the organization in the short to medium term. The highly structured information provided by these systems allows managers to evaluate an organization's performance by comparing current with previous outputs.

Functions of a MIS

MIS are built on the data provided by the TPS

Functions of a MIS in terms of data processing requirements		
Inputs	Processing	Outputs
Internal Transactions Internal Files Structured data	Sorting Merging Summarizing	Summary reports Action reports Detailed reports

Some examples of MIS

- Sales management systems
- Inventory control systems
- Budgeting systems
- Management Reporting Systems (MRS)
- Personnel (HRM) systems

The role of MIS

- Based on internal information flows
- Support relatively structured decisions
- Inflexible and have little analytical capacity
- Used by lower and middle managerial levels
- Deals with the past and present rather than the future
- Efficiency oriented?

- **Decision support systems**

A Decision Support System can be seen as a knowledge based system, used by senior managers, which facilitates the creation of knowledge and allow its integration into the organization. These systems are often used to analyze existing structured information and allow managers to project the potential effects of their decisions into the future. Such systems are usually interactive and are used to solve ill structured problems. They offer access to databases, analytical tools, allow "what if" simulations, and may support the exchange of information within the organization.

Functions of a DSS

DSS manipulate and build upon the information from a MIS and/or TPS to generate insights and new information.

Functions of a DSS in terms of data processing requirements		
Inputs	Processing	Outputs
Internal Transactions Internal Files External Information?	Modelling Simulation Analysis Summarizing	Summary reports Forecasts Graphs / Plots

Some examples of DSS

- Group Decision Support Systems (GDSS)
- Computer Supported Co-operative work (CSCW)
- Logistics systems
- Financial Planning systems
- Spreadsheet Models?

The role of DSS

- Support ill- structured or semi-structured decisions
- Have analytical and/or modelling capacity
- Used by more senior managerial levels
- Are concerned with predicting the future
- Are effectiveness oriented?

- **Expert systems**

- **Office automation systems**

The **office automation systems,** also called OAS consist of applications designed to help the daily work of the administration of an organization, part of this type of software are the word processors , the leaves calculation , the editors of presentations , customer email , etc.. When several of these applications are grouped into a single software package for easy distribution and installation, the set is known by the name of office suite .

Perhaps the most popular software package that can fit the definition of OAS (and to the office suite) is Microsoft Office in all its versions. This software, part of the company Microsoft , officially operates under the operating systems Microsoft Windows and Apple Mac OS , although it does in Linux when using emulators.

There are other office suites available for any user to be distributed freely, some of which are:

Today, with the emergence of the philosophy of Web 2.0 office suite are proliferating **online**, which are nothing more than applications that perform the same functions as the OAS classic desktop, but available for use in any portal Internet. These suites have the advantage that a user can work with your own documents from any computer connected to the Internet, also in these systems is usually very easy to share documents, facilitating collaborative work.

- **Others**

2. Executive Information Systems

What is an EIS?

Executive Information Systems are strategic-level information systems that are found at the top of the Pyramid. They help executives and senior managers analyze the environment in which the organization operates, to identify long-term trends, and to plan appropriate courses of action. The information in such systems is often weakly structured and comes from both internal and external sources. Executive Information System are designed to be operated directly by executives without the need for intermediaries and easily tailored to the preferences of the individual using them.

Functions of an EIS

EIS organizes and presents data and information from both external data sources and internal MIS or TPS in order to support and extend the inherent capabilities of senior executives.

Functions of a EIS in terms of data processing requirements		
Inputs	Processing	Outputs
External Data Internal Files	Summarizing Simulation	Summary reports Forecasts

Pre-defined models	"Drilling Down"	Graphs / Plots

Some examples of EIS

Executive Information Systems tend to be highly individualized and are often custom made for a particular client group; however, a number of off-the-shelf EIS packages do exist and many enterprise level systems offer a customizable EIS module.

The role of EIS

- Are concerned with ease of use
- Are concerned with predicting the future
- Are effectiveness oriented
- Are highly flexible
- Support unstructured decisions
- Use internal and external data sources
- Used only at the most senior management levels

Roles of information systems stake holders

- **systems owners**

Are usually the individuals with budgetary control over the system. This person should be able to authorize purchases of software and / or hardware required to meet policy requirements, and / or have the authority to decommission the system.

The system owners own the systems because they have invested in the assets. They are responsible for them and recognize their potential. They keep your customers happy and impact your bottom line. Of course you want to maximize their value. To do that, you need to monitor performance efficiency and maintenance events. You need to track and manage your assets to realize total compliance and positive investment outcomes.

- **System Users are classified into two categories namely:**
- Internal System Users
-
- Clerical and Service workers
- Technical and professional staffs
- Supervisors, Middle managers and executive managers
-

- External System Users comprises of the following:
- Customers
- Suppliers
- Partners

- The end users of the system refer to the people who use computers to perform their jobs, like desktop operators. Further, end users can be divided into various categories.

- Very first users are the hands-on users, who actually interact with the system. They are the people who feed in the input data and get output data.

- Other users are the indirect end users who do not interact with the systems hardware and software. However, these users benefit from the results of these systems. These types of users can be managers of organization using that system.
- There are third types of users who have management responsibilities for application systems. They oversee investment in the development or use of the system.

- Fourth types of users are senior managers. They are responsible for evaluating organization's exposure to risk from the systems failure.

System Analysts

1. Systems Analysts Collaborate with IT Professionals All Throughout the SDLC

Systems analysts certainly aren‘t limited to working just on software development projects. Yet, for some systems analysts, this is all they work on. At every stage of the systems development life cycle (SDLC), they team up with programmers, system designers, QA testers, system intergrators and so forth, to build systems.

2. Systems Analysts Document Systems

Because they collaborate with professionals across the entire SLDC, systems analysts often serve as information brokers. The documentation prepared by systems analysts often become the final word on what a system is or does.

Systems analysts document IT systems to understand, change, improve, and help build these systems. Broadly, they document what is happening in current systems. But they also document what can be expected in systems that haven‘t been built yet. This work is often done in collaboration with technical writers, systems designers, and systems architects. Typical things that systems analysts document include:

- User scenarios
- Functional activities
- Classes
- Data flows
- Interfaces between systems

Examples of documents that systems analysts author or co-author include:

- Requirements documents
- Feasibility studies
- Design specifications
- Software development kits (SDKs)
- Flowcharts and diagrams
- Training documents
- Testing documents
- Disaster recovery plans (DRPs)

3. Systems Analysts Gather and Analyze Requirements

Information overload occasionally happens to each of us. This is where systems analysts come in handy. They transform a seemingly incomprehensible avalanche of information into something useful and anything that will translate business goals into technical solutions. Here are some examples of questions that systems analysts can ask during the requirements phase of a project:

Objectives

- What do you want the system do?
- What is in and out of scope for this project?
- When do you want the system to go live?
- How much money in the budget is there for systems analysis?

Problems

- What problems will the proposed new system fix?
- What problems do you want fixed that relate to the existing system?
- Is there a log of all software bugs?
- Have the bugs been prioritized and ranked?
- What bug tracking software do you use?

Changes

- What kinds of changes are needed?
- Are the changes classified as modifications or enhancements?

There are many types of requirements that a systems analyst will collect and analyze. Three of the main ones are:

- **Business Requirements.** High level requirements that managers and directors would typically understand.
- **Functional Requirements.** Detailed requirements that specify exactly what needs to be delivered. These are generally read by business analysts, software engineers, and project managers.
- **Technical Requirements.** Detailed requirements about how the system is built, including which language it will be programmed in, what standards are to be followed, etc. Technical requirements are often read by software architects and developers.

4. Systems Analysts Choose, Upgrade, and Configure Software

Clearly, not all software is custom-built. There are thousands of applications that are prepackaged. These kinds of programs can be customized by setting various preferences, yet they can‘t be as individualized as custom-programmed software.

Systems analysts often play a crucial role in both picking and configuring software. Here are some examples of tasks that systems analysts do in conjunction with system architects and system designers:

Software Selection

- Inviting software vendors to submit request for proposals (RFPs).
- Vetting vendors.
- Reviewing systems.
- Determining which system most fits your requirements.
- Recommending which system to implement.

Software Configuration

- Working with the system administrator to install the software.
- Assigning roles and privileges to users.

5. Systems Analysts Work with Tons of People

To get the job done, systems analysts have to work with a large variety of people, on top of the IT professionals they always collaborate with. Here are a few of them:

- **Sponsors** are the people who want a system to be built so they can make money from it. In business speak: systems analysts work with sponsors to assess whether the financial investment in a system will be recouped by gains in efficiency and effectiveness.
- Systems analysts often work with **programmers** who write code to modify existing systems and/or build new ones.

- Systems analysts work with **technical writers** to draft user guides, technical overviews, instruction manuals, and frequently asked questions (FAQs).
- Systems analysts work with **Quality Assurance (QA) analysts** to test systems to ensure that critical requirements are met.

- **systems designers: are the people concerned with the design of systems and includes the following:**

  - Database Administrators
  - Network Architects
  - Web Architects
  - Graphics Artists
  - Security Experts
  - Technology specialists

Systems developers:

These are the people who create and write computer programs. Some develop the applications programs that allow people to do specific tasks on a computer or other device. Others develop the underlying systems that run the devices or control networks

- **Other**

Computer programmers

Computer programmers write code to create software programs. They turn the program designs created by software developers and engineers into instructions that a computer can follow.

System Builders

- Application programmers (writes application programs)
- System programmers (writes operating systems programs)
- Databases Programmers
- Network Administrators
- Security Administrators
- Webmasters (manages web pages)
- Software Integrators (they integrate different software and have one working system)

Computer network architects:

Design and build data communication networks, including local area networks (LANs), wide area networks (WANs), and intranets. These networks range from a small connection between two offices to a multinational series of globally distributed communications systems.

Computer support specialists:

Provide help and advice to people and organizations using computer software or equipment. Some, called computer network support specialists, support information technology (IT) employees within their organization. Others, called computer user support specialists, assist non-IT users who are having computer problems.

Database administrators (DBAs):

Use specialized software to store and organize data, such as financial information and customer shipping records. They make sure that data are available to users and are secure from unauthorized access.

Computer networks:

Are critical parts of almost every organization. Network and computer systems administrators are responsible for the day-to-day operation of these networks.

Web developers: These are the people who design and create websites

They are responsible for the look of the site. They are also responsible for the site‘s technical aspects, such as performance and capacity, which are measures of a website‘s speed and how much traffic the site can handle. They also may create content for the site.

TOPIC 2: SYSTEMS THEORY/CONCEPT

THEORY

16.2.2.2. T0 Specific Objectives

By the end of this topic, the trainee should be able to:

a) Explain systems concept

b) Describe the components of a system

c) Describe the classification of systems

d) Explain system properties

e) Describe the types of systems

CONTENT

Systems theory/concept.

Systems theory was introduced by biologist L. von Bertalanffy in the 1930s as a modeling devise that accommodates the interrelationships and overlap between separate disciplines. The reality is that when scientists and philosophers first tried to explain how things worked in the universe, there were no separate disciplines. There were simply questions to be answered. But as we started understanding more and more, the sciences broke down into chemistry, physics, biology, and then biophysics, biochemistry, physical chemistry, etc. so that related components of a problem were investigated in isolation from one another. The Systems Theory introduced by von Bertalanffy reminds us of the value of integration of parts of a problem. Problems cannot be solved as well if they are considered in isolation from interrelated components. An enormous advantage systems analysts have in knowing the definitions of systems theory is that they present us with ideal guidelines for our initial familiarization with a new problem, which of course is a new system.

Systems Theory Terms/PRICIPLES

Problem

A problem can be a question looking for an answer, a situation (such as an existing information system) that isn't working properly and needs improving, or a new opportunity or idea that is worthy of further consideration. In other words, when we speak of a "problem" in systems analysis and design, we don't necessarily mean that there is something wrong. We mean that there is a situation that needs to be understood and a solution to be determined.

System

A system is a set of *related components* that *work together* in a particular *environment* to perform whatever *functions* are required to achieve the system's *objective*.

Goal Seeking

A system is goal-seeking by definition. When the definition of a system says that a system's components work together to achieve a common objective it means that the system seeks to complete a goal. For example, the objective of the digestive system is ensure that food is digested, with some byproducts going into the related circulatory system to nurture the body and other byproducts being expelled. The objective of a payroll system is likely to be to produce complete, correct and timely output in the form of cheques, reports, and updated history files. *It is important to be able to identify the objectives of any existing or new system to be able to understand it and evaluate its effectiveness.* In an information system, the components include people, procedures, data, software, and hardware.

Entropy

Entropy is a measure of the degree of disorder in a system. It is a familiar term in thermodynamics, when considering chemical systems, and is also relevant to information systems. The concept of entropy says that any system will tend towards disorder. Knowing that, we can put checks in place to monitor the correctness of the output of a system.

Internal Environment

A system operates in an environment with both internal and external components. Its internal environment is that part of its environment over which it has some control. If some aspect of the internal environment is causing some difficulty for the system, that aspect can be altered. For example, a particular information system operates in a particular office environment. If a requirement of the information system is that its users must collect data that hasn't been collected previously, this new activity can be asked of them.

External Environment

A system's external environment is that part of its environment over which it has no control, but it still affects the requirements of the system. For example, in a payroll system, the revenue authority tax laws, the NHIF laws affect the procedures in the system. The tax laws must be reflected in the system, and if the laws change, the system must change to accommodate those changes. *So an analyst must be aware of the requirements of both the internal and external environments in which an information system will work.*

Subsystem

A system is usually composed of self-contained but interrelated systems that are called subsystems. It is important to be able to recognise these subsystems, because understanding this interdependence is vital to developing a *complete* system.

Supersystem

A system composed of two or more systems may be called a supersystem of those systems.

System Boundary

A system boundary may be thought of as the point at which data flows (perhaps as output) from one system to another (perhaps as input). The degree to which data is free to flow from one system to another is known as the permeability of the boundary. A permeable boundary allows data to flow freely, resulting in an open system. An impermeable boundary is one which strictly controls (or even restricts) the acceptance or dispensing of data, resulting in a closed system.

Interdependence

One of the most important concepts in Systems Theory is the notion of interdependence between systems (or subsystems). Systems rarely exist in isolation. For example, a payroll system has to access and update a personnel system. It is important for an analyst to identify these interdependence early. It may be the case that changes you make to one system will affect another in ways you haven't considered, or vice versa.

Components/elements of a system

- **input**

Input is *anything we wish to embed in a system for some type of use*. A variety of sources are used to input: keyboard, scanner, microphone, mouse, even another computer. What we input has a purpose - but until it is processed and generated in some form of output, it doesn't do us much good.

- **processing**

Processing takes place in the internal parts of the computer. It is the *act of taking inputted data and converting it to something usable*. What we typically see on the screen in today's computer world (known as ***w**hat **y**ou **s**ee **i**s **w**hat **y**ou **g**et* or **WYSIWYG**) is the result of our input being processed by some program so we can have usable output:

Output

Output, or processed information in a usable format, comes in many different forms: monitor or printer for visual work, a speaker for audio. Sometimes our output is short-term, such as printing a photo, and sometimes what we work on needs to be kept around for a while. That's where storage comes in

Storage is the term used to indicate we will be *saving data for a period of time*. We store for many reasons: for future reference; to prevent full loss of data; But, storage is vital. There are several mediums on which we can keep output and processed data: a hard disk, a USB drive, a CD.

Feedback

To be effective and efficient a system needs a feedback mechanism that can ascertain whether the outputs of the system are what they should be. If not. a system should have the ability to adjust its inputs or processes to improve the outputs. An ideal system is self-regulating. The feedback mechanism in an information system may be automated or may be manual.

Types of systems

- **Man made Systems**

As we have previously defined systems, a number of these systems are constructed, organized, and maintained by humans. These man made systems include such things as:

- Social system: include organizations of laws, doctrines, customers and so on.
- Transportation systems: that includes networks of highways, canals, airlines, oceans tankers etc.
- Communication systems: includes telephone, telex, smoke signals, the hand signals used by stock market traders and so on
- Manufacturing systems: where we have factories, assembly lines etc.
- Financial systems: that include accounting, inventory, general ledger, stock brokerage.

Most of these systems include computers today, indeed many could not survive without computers. However it is equally important to point out that such systems existed before there were computers, some of these systems are still completely non-computerised and may remain that way for many years to come. As a system analyst you will naturally assume that every system that you come in contact with should be computerised, and the customer or the user (the owner of the system) whom you interact with will generally assume that you have such bias.

Why should some information systems not be automated?, there may be many reasons, but here are some of the more common ones:

- Cost: it may be cheaper to continue carrying out the system functions and storing the systems information manually.
- Convenience: an automated system may take up too much room, make too much noise, generate too much heat, or consume too much electricity etc.
- Security: if the information system is maintaining sensitive, confidential data, the user may not feel that an automated system is sufficiently secure. The user may want the ability to keep the information physically protected and locked up.
- Maintainability: the user might argue that a computerized information system would be cost-effective except that there is nobody on the staff that can maintain the computer hardware and software, nobody would repair the system if its broken down.

- **Automated System**

Automated systems are actually man made systems that interact with or are controlled by one or more computers.. no doubt you have seen many different examples of automated systems in your day to day life. It seems that every aspect of our modern society is computerized. As a result we can distinguish many different kinds of automated systems.

Even though there are many different kinds of automated system, they all tend to have common components. And these components include:

- ➢ Computer hardware- CPUs, disks, printers, magnetic tape drives etc.
- ➢ Computer software- system programs such as operating systems, database systems, and telecommunication control programs, and application programs that carry out the functions that the user wants.
- ➢ People- those who operate the system.
- ➢ Data- the information that the system remembers over a period of time
- ➢ Procedures- these are the rules and formal policies and instructions for operating the system

A more useful categorization of automated systems include:

- On-line systems
- Real time systems
- Decision support systems
- Knowledge based systems.

Classification of systems

- **open Vs closed**

  An open system is one that interacts with its environment and thus exchanges information, material, or energy with the environment, including random and undefined inputs. Open systems are adaptive in nature as they tend to react with the environment in such a way organizing', in the sense that they change their continued existence. Such systems are _self-organizing‘, because they change their organization in response to changing conditions.

  A closed system is one, which doesn‘t interact with its environment. Such systems, in business world, are rare. Thus the systems that are relatively isolated from the environment but not completely closed are termed closed systems.
- **adaptive**

  A system is said to be adaptive if it modifies itself with the changes in its environment.

  A democratic system of government is an example of adaptive system as it changes to accommodate the changes in the environment.

  A non-adaptive system does not react to changes in its environment.

  - An autocratic system of governance is an example of non-adaptive system. It does not change or adapt to changes in the environment.
- **deterministic**

  A deterministic system is one in which the occurrence of all events is known with certainty. If the description of the system state at a particular point of time of its operation is given, the next state can be perfectly predicted.
- **Probabilistic**

A probabilistic system is one in which the occurrence of events cannot be perfectly predicted. Though the behaviour of such a system can be described in terms of probability, a certain degree of error is always attached to the prediction of the behaviour of the system.

Classification of properties

- **HARD PROPERTIES**

  Most business or organisational problems can be defined in terms of information with both hard and soft properties. Whatever sector we work in, some of the issues we face can be precisely measured, for example the cost of an item, the strength of a particular piece of material, or the number of employees in our organisation – and the answer is not conditioned by the measurer‗s sense of value. This data is said to have *hard properties*.

Soft Properties

- On the other hand, some issues are not capable of such precise measurement, and the assessment contains at least an element of judgement or is subjective in some way. Information such as whether a material looks attractive, whether we should recruit a particular person or what the average rate of price inflation will be over the next 10 years is subjective and we say that it has *soft properties.*

- Needless to say these definitions can get blurred. For example, when we are measuring absences from the workplace, whether someone is absent or not would appear to be clear cut and not a matter of judgement, so we might be inclined to consider this information as having hard properties.

TOPIC 3: SYSTEMS DEVELOPMENT LIFE CYCLE (SDLC)

THEORY

16.2.2.3. T0 Specific Objectives

By the end of this topic, the trainee should be able to:

a) explain the meaning of SDLC

b) describe SDLC stages

CONTENT

Meaning of SDLC

SDLC stages

- **problem definition**
- **feasibility study**
- **systems analysis**
- **systems design and development**
- **implementation**
- **maintenance and review**

The Systems Development Cycle:

The systems approach can be applied to the solution of many types of problems. When this involves the development of information system solutions to business problems, it is called information systems development or application development. Most computer-based information systems are conceived, designed, and implemented using some form of systematic development process. In this process, end users and information specialists design information systems based on an analysis of the information requirements of an organization. Thus, a major part of this process is known as systems analysis and design.

When the systems approach is applied to the development of information system solutions, a multistep process or cycle emerges. This is frequently called the information systems development cycle, also known as the systems development life cycle (SDLC).

Steps involved and products produced in the traditional information systems development cycle:

1. Systems investigation - Product: Feasibility Study report
2. Systems analysis - Product: Functional Requirements
3. Systems design - Product: Systems Specifications
4. Systems implementation - Product: Operational System
5. Systems maintenance - Product: Improved System

1. All the activities involved are highly related and interdependent.
2. Several developmental activities can occur at the same time.
3. Different parts of a development project can be at different stages of the development cycle.
4. Analysts may recycle back at any time to repeat previous activities in order to modify and improve a system being developed.

5. Developments such as computer-aided systems and end user development are automating and changing some of the activities of information systems development. These developments are improving the quality of systems development and making it easier for IS professionals, while enabling more end users to develop their own systems.

Starting the Systems Development Process:

The first step in the systems development process is the systems investigation stage. This step may involve consideration of proposals generated by an information systems planning process. The investigation stage also includes the preliminary study of proposed information system solutions to end user business problems.

The three steps of the systems investigation stage involve:

1. Determine whether a business problem or opportunity exists.

Problems definition and Opportunities:

To solve a problem or pursue an opportunity requires a thorough understanding of the situation at hand. This requires separating problems from symptoms, determining objectives and constraints, and, more important, viewing the problem or opportunity in a systems context.

- Problem: - is a basic condition that is causing undesirable results.
- Opportunity: - is a basic condition that presents the potential for desirable results.
- Symptoms: - are merely signals of an underlying cause or problem.

2. Conduct a feasibility study to determine whether a new or improved information system is a feasible solution

3. Develop a project management plan and obtain management approval.

Feasibility Studies

Because the process of developing a major information system can be costly, the systems investigation stage frequently requires a preliminary study called a feasibility study. A feasibility study is a preliminary study which investigates the information needs of prospective users and determine the resource requirements, cost, benefits, and feasibility of a proposed project.

Steps of a feasibility study:

1. Gather information/data for a feasibility study.
2. Formalize a written report including the preliminary specifications and a developmental plan for the proposed system.
3. Submit the report management for approval.
4. Begin system analysis (if management approves the recommendations of the feasibility study).

The goal of feasibility studies is to:

1. Evaluate alternative systems
2. Propose the most feasible and desirable systems for development.

Feasibility of a system can be evaluated in terms of four major categories:

1. Organizational Feasibility - focuses on how well a proposed information system supports the objectives of the organization and its strategic plan for information systems.

2. Economic Feasibility - focuses on whether the tangible costs and benefits of the proposed system will exceed the costs of developing and operating it.
3. Technical Feasibility - focuses on the reliable/capabilities of the hardware and software to meet the needs of the proposed system, and whether they can be acquired or developed in the required time.

4. Operation Feasibility - focuses on the willingness and ability of the management, employees, customers, suppliers, and others to operate, use, and support the proposed system.

Cost/Benefit Analysis

Every legitimate solution will have some advantages or benefits, and some disadvantages or costs. These advantages and disadvantages are identified when each alternative solution is evaluated. This process is typically called cost/benefit analysis.

Tangible Costs - are costs and benefits that can be quantified (e.g., cost of hardware and software, employee salaries, and other quantifiable costs needed to develop and implement a solution).

Intangible Costs - costs and benefits that cannot be quantified (e.g., loss of customer goodwill or employee morale caused by errors and disruptions arising from the installation of a new system).

Tangible Benefits - are favourable results (e.g., decrease in payroll costs caused by a reduction in personnel or a decrease in inventory carrying costs caused by a reduction in inventory)

Intangible Benefits - are hard to estimate (e.g., better customer service or faster and more accurate information for management).

System Analysis

Systems analysis is an in-depth study of end user information needs which produces functional requirements that are used as the basis for the design of a new information system. System analysis traditionally involves a detailed study of:

1. The information needs of the organization and the end users.
2. The activities, resources, and products of any present information systems
3. The information systems capabilities required to meet the information needs of end users.

Organizational Analysis

Organizational analysis involves evaluating the organizational and environmental systems and subsystems involved in any situation. Systems analysis traditionally involves a detailed study of the organizations:

1. Environment
2. Management structure
3. People
4. Business activities
5. Environmental systems it deals with
6. Current information systems

Analysis of the Present System

Before designing a new system, a detailed analysis of the current system (manual or automated) must be completed. An analysis of the present system involves analysing activities, resources, and the products. You must analyse how the present system uses:

1. Hardware, software, people resources to convert data resources into information products, such as reports and displays.
2. Document how the information activities if input, processing, output, storage, and control are being accomplished.

Functional Requirements Analysis

This step of the systems analysis is one of the most difficult. Steps involve:

1. Determining specific information needs
2. Determining the information processing capabilities required for each system activity (input, processing, output, storage, and control) to meet the needs. Goal is to identify What should be done NOT how to do it.
3. Develop functional requirements (information requirements that are not tied to the hardware, software, and people resources that end users presently use or might use in the new system).

Systems Design

System analysis describes what a system should do to meet the information needs of users. System design specifies how the system will accomplish this objective. Systems design consists of design activities, which produce systems specifications satisfying the functional requirements developed in the systems analysis stage. These specifications are used as the basis for:

1. Software development
2. Hardware acquisition
3. System testing
4. Other activities of the implementation stage.

User Interface, Data, and Process Design

The systems design concept focuses on three major products or deliverables, that should result from the design stage. System design consists of three activities:

1. User Interface Design
2. Data Design
3. Process Design

User Interface Design:

User interface design focuses on supporting the interactions between end users and their computer-based applications. Designers concentrate on:

- The design of attractive and efficient forms of user input and output, such as easy-to-use Internet or intranet web pages
- Designing methods of converting human-readable documents to machine-readable input, such as optical scanning of business forms.
- Design tips to keep in mind:
  - Keep it simple
  - Keep it clean
  - Organize logically

User interface design is frequently a prototyping process, where working models or prototypes of user interface methods are designed and modified with feedback from end users. User interface design produces detailed specifications for information products such as:

1. Display screens
2. Interactive user/computer dialogues
3. Audio responses
4. Forms
5. Documents

6. Reports.

Data Design

The data design activity focuses on the design of the structure of databases and files to be used by a proposed information system. Data design frequently produces a data dictionary, which catalogues detailed descriptions of the:

1. Attributes or characteristics of the entities (objects, people, places, events) about which the proposed information system needs to maintain information.
2. Relationships these entities have to each other.
3. Specific data elements (databases, files, records, etc.) that need to be maintained for each entity tracked by the information system
4. Integrity rules that govern how each data element is specified and used in the information system.

Process Design

The process design activity focuses on the design of software resources, that is, computer programs and of procedures needed by the proposed information system. It concentrates on developing detailed specifications for the program modules that will have to be purchased as software packages or developed by custom programming. Process design produces:

1. Detailed program specifications and procedures needed to meet the user interface and data design specifications that are developed.
2. Produces specifications that meet the functional control and performance requirements developed in the analysis stage.

System Specifications

System specification focuses on defining the systems specifications required for the proposed information system. Typically, it specifies:

1. Hardware resources (machines and media)
2. Software resources (programs and procedures)
3. Network resources (communications media and networks)
3. People resources (end users & information systems staff).
4. How resources will be used to convert data resources (stored in files and databases they design) into information products (displays, responses, reports, and documents).

System Development-Implementing a New Information System:

Once a proposed information system has been designed, it must be implemented. The systems implementation stage involves:

1. Hardware and software acquisition
2. Software development
3. Testing of programs and procedures
4. Development of documentation
5. Installation activities
6. Education and training of end users and specialists who will operate the new system.
7. Converting from the use of the present system to the operation of a new or improved system.

   Converting to a new system may involve:

   Parallel System - Operating both a new system and an old system at the same time for a trial period.

   Pilot System - Operate a pilot system on a trial basis at one location.

Phasing - Phasing in the new system one application or location at a time.
Plunge (Cutover) - Converting immediately to the new system.

Maintenance of Information Systems

Systems maintenance is the final stage of the systems development cycle. It involves the monitoring, evaluating, and modifying of a system to make desirable or necessary improvements. This may include:

1. Post implementation review process to ensure that the new system meets the objectives established for it.
2. Error detected in the development or use of the system are corrected.
3. Later modifications to a system may also become necessary due to changes within the business or the business environment.

TOPIC 4: PROBLEM DEFINITION

THEORY

16.2.2.4. T0 Specific Objectives

By the end of this topic, the trainee should be able to:

a) identify problem indicators

b) describe the contents of a TOR

CONTENT

Problem Indicators

Contents of A Terms of Reference (TOR)

A ToR is a formal document, but it is typically not very long. It can be used to describe a project before a full project charter (or project initiation document) is produced. It is more typically used to set out the terms of reference for a particular work stream or sub-project, and is written by the project manager with input from the functional lead who is managing that section of the work. You could also produce one for a phase of the project, for example, the initial scoping phase. In short, a ToR can cover many things but generally sets out the scope of what needs to be done on a particular piece of work.
The ToR document includes:

Background

The context for the work, the overall aims of the work and any references to other pieces of work that the team should take into account when commencing the work.

Objectives

What is this piece of work going to achieve? What problem is it going to solve? These should be set out in a way that everyone can understand, avoiding jargon.

Scope

This section briefly covers what is in scope and out of scope of the work. It is easiest to list this in bullet point format, as if more detail is required this can be produced in a full requirements document. Bullet points in this section should cover areas like:

- The technical systems involved or that are required
- The business processes that will be affected by this work
- What hardware and software is required (or specifically out of scope)
- Where the project will take place, what locations are affected and what locations will be out of scope for the purpose of this work
- What third parties will be involved
- Who will be affected, and which teams or individuals will specifically be out of scope.

You will probably think of other things to include in the scope. A good tip is to bring the team together (if you don‘t have a full team together at this point bring together some colleagues).

Constraints

This short section documents any project constraints, such as timescales, the available budget, the resources available or any legislative or regulatory frameworks that have to be considered.

Assumptions

Include any assumptions in the ToR. These are thing that you don‘t yet know for certain but that will have an impact on the piece of work later on. If you are updating your ToR later you can update this section as you may well have been able to ratify your assumptions by then.

Roles and responsibilities

Who is on the team? In this section document the names and roles of the people who will be carrying out the work. You can also include their availability, such as if they are only available to work on the project two days a week. Also include details of who is sponsoring the work. If it is a full project, this will be the project sponsor. If it is a workstream or part of an existing project, the person to whom the workstream lead reports (probably you, the project manager) is the right person to mention here.

Deliverables

What is the work going to deliver? Make a list – it doesn‘t have to be too detailed at this point as all you really need is a high level description of the outputs of the work. You could include a screenshot of the high level milestones from your project plan here too.

Format

Ideally, you should aim to get your ToR on a couple of pages. There is no need for a fancy cover page or appendices. Put a header and footer on the document and get started! You can always include version control information in a short table at the beginning or end.

Other uses for a ToR

You can also set out a ToR for meetings, so your Project Board or Steering Group may have a ToR that explains what they are there to do and how they will go about conducting the business of governance or oversight on the project. You could also have one for other types of meetings, such as a terms of reference for risk management meetings. As long as the ToR clearly sets out the scope of an activity and explains what is also out of scope, then it will do its job.

TOPIC 5: FEASIBILITY STUDY

THEORY

CONTENT

Introduction

The process of analyzing whether the proposal is feasible or not is called feasibility analysis. if it is not feasible, we need to look after other alternatives. Feasibility study mainly focuses on the demand of the system that affects the overall development of the information system.

It is an assessment of the practicality of a proposed plan or method. Which helps to find the strengths and weaknesses of an existing business or proposed venture, opportunities and threats present in the environment, the resources required to carry through, and ultimately the prospects for success.

Feasibility study is used for:

1. To determine whether the objectives stated in the assignment brief are reasonably attainable within the limitation and financial constraints period.
2. To define major problem areas, so that the system analyst can plan the strategy for the field investigation.
3. To find areas where potential exists for making saving in money, time or effort.
4. To determine the approximate time required for the full investigation and cost.
5. To discover the areas where some specialist knowledge needed for the full investigation.

There are different types of feasibility study... which are includes:,

- Technical feasibility
- Economical feasibility
- Operational feasibility
- Schedule feasibility

Technical feasibility

It measure of the availability of technical resources (hardware components or technical equipment). It also studies the availability of the technical manpower for the project. [if the work performances of the technical manpower are not experienced, the entire system will be certainly insufficient.]

Economical feasibility

Economical feasibility measures whether finances (investments) are available for proposed solution, i.e. it looks at the financial aspects (cost/ effectiveness) of the project. This is often called a cost-benefit analysis.

Operational feasibility

It is a measure of how well the solution of problems or a specific alternative solution will work in the organization. It is also measure of how people feel about the system. If the system is not easy to operate, than operational process would be difficult. The operator of the system should be given proper training. The system should be made such that the user can interface the system without any problem.

Schedule feasibility

If a deadline (time-limit) is established, it is called schedule feasibility, i.e the deadline of the project is studied under the scheduled feasibility. The scheduled feasibility is also depends upon available manpower and economical condition as well.

□ **Legal/Ethical Feasibility** - What are the legal implications of the project? What sort of ethical considerations are there? You need to make sure that any project undertaken will meet all legal and ethical requirements before the project is on the table.

□ **Resource Feasibility** - Do you have enough resources, what resources will be required, what facilities will be required for the project, etc.

Fact finding methods

This involves collection of information about the existing system on which to base analysis in order to determine whether users current needs are being met.

The following are some of the activities that are involved

1. **Functional requirement.** The requirement should be established.
2. **Determination of the proposed system requirement**. This is necessary it may suggest a change in the existing system requirement.
3. Establish any weakness or problems associated with the present system working methods and procedures
4. Determination of the organizational growth rate, will assist in determination of the growth of the volume of transactions to be processed.
5. Determination of the organizations structure objective and the cost associated with the present system

Fact finding comprises of the following:

- ✓ Fact gathering
- ✓ Fact recording
- ✓ Fact evaluation

Data gathering tools/techniques/methods

The widely used methods for data collection include the following:

a) Questionnaires
b) Interviews
c) Observations
d) Records inspection/ document reviews
e) Sampling

A. Questionnaires

Questionnaire is a special document that allows the analyst to ask a number of standard prepared questions set to be asked to a large number people in order to gather information from them.

Suitability:

It is suitable to use when:

- The system analyst is located at a considerable long distance from the respondents
- There is a large number of respondents such that interviewing them will be limited by time
- The questions to be asked are simple and straight forward and require direct answers
- Limited information is required from a large number of people
- Used as a means to verify facts found using other methods

Advantages of Using Questionnaires

I. They provide a cheap means of gathering information from a large number of people.
II. They encourage individual to provide response without fear, intimidation or victimization
III. The respondents can complete the questionnaires at their own convenient time with minimal interruption from their work
IV. Questionnaires are presented consistently to all participating without bias.

Disadvantages

I. Response is often too slow since the respondents complete and return the forms at their own convenience
II. They don‘t provide an opportunity for respondents to obtain clarification of questions which may appear vague or ambiguous
III. The design of questionnaires require an expert who may charge expensively and may not be economical when administered to a small group of respondents
IV. All forms may not be returned and not all questions may be answered which leads to incomplete data for analysis

Requirements for preparing a questionnaire

- Questions should be simple and clear
- Forms should be neat
- Questions should be logically organised
- Questions should be objectively oriented and should avoid leading questions

B. Interviews

This is a face to face conversation between the analyst (**the interviewer**) and the users (**interviewee**). The analysts will obtain answers to questions he asks the interviewee. The interviewee will give suggestions and recommendations that may assist during the design of the proposed system.

Purpose for the interviews

i. They act as a method of fact finding to gather information or responses about the existing system
ii. Used for verifying facts gathered through other methods
iii. Used to get the user involved in the development of the new system

Interviews are used in the following circumstances

i. When respondents are few
ii. When respondents are physically available and accessible
iii. When immediate response is required
iv. When the analyst wishes to verify validity of facts collected through other methods
v. When the analyst wishes to seek direct answers, opinions, suggestions and other detailed information regarding the system to be developed.

Advantages of interviews

i. The response rate tends to be high and immediate
ii. Detailed facts can be obtained from individual respondents
iii. They are a more powerful means of getting information since there is direct contact with the respondent (body language can easily be seen)
iv. Analyst can frame questions directly to individual depending on their level of understanding allowing facts to be obtained.

Disadvantages

i. They are costly and time consuming when large groups are involved
ii. Success depends highly on the analysts competence, human relations skills and experience
iii. The respondents may feel that they are being grilled

C. Observations

This is the most effective fact finding technique but requires the analyst to participate in performing some activities carried out by the user. The analyst may choose to watch users as they perform their activities and gather the facts intended

Circumstances that require observation

- When validity of facts gathered through other methods are questionable
- When complexity of certain aspects of a system prevent a clear explanation by the respondents or the user

- Used to confirm that procedures specified in the manuals are being followed in the case of an existing system
- When there is need to obtain first hand and reliable information

Advantages

- Data gathered is highly reliable thus the method can be used to verify facts collected through other methods
- There is an opportunity for the analyst to see what happens exactly including the tasks which are hard to explain clearly in words.
- In accurately described tasks can easily be identified
- Relatively cheap compared to other methods

Disadvantages

- People always feel uncomfortable when being observed and may behave abnormally thus influencing the analyst conclusion.
- The exercise may take place at odd times thus inconveniencing those involved
- The analyst may observe exceptional activities leaving some critical areas.

Class Discussion group

Research on the following data collection methods

1. **Document review/Records inspection and**
2. **Sampling method and state their advantages and disadvantages**

TOPIC 6: SYSTEMS ANALYSIS

Sub topic: Meaning and importance of systems analysis

Systems analysis is an in-depth study ***of end user information needs*** which produces functional requirements that are used as the basis for the design of a new information system.

Sub topic: Importance of Systems analysis

The importance of system analysis traditionally involves a detailed study and determination of the following details:

1. The determination of information needs of the organization and the end users.

2. The analyst will be able to identify activities, resources, and products of any present / current information systems

3. The analysis will reveal information systems capabilities required to meet the information needs of end users.

Sub topic: Systems analysis approaches/methods

- **Model –driven:** there are mainly four approaches used in model driven namely:

○ **Modern structured design**

A system design technique that decomposes the system's processes into manageable components.

- Synonyms (although technically inaccurate) are top-down program design and structured programming.
- Design in a top-down hierarchy of modules
- Easier to implement and maintain (change).
- Modules should be highly cohesive
- Accomplish one function only
- Modules should be loosely coupled
- Minimally dependent on one another

○ **Information engineering (IE)**

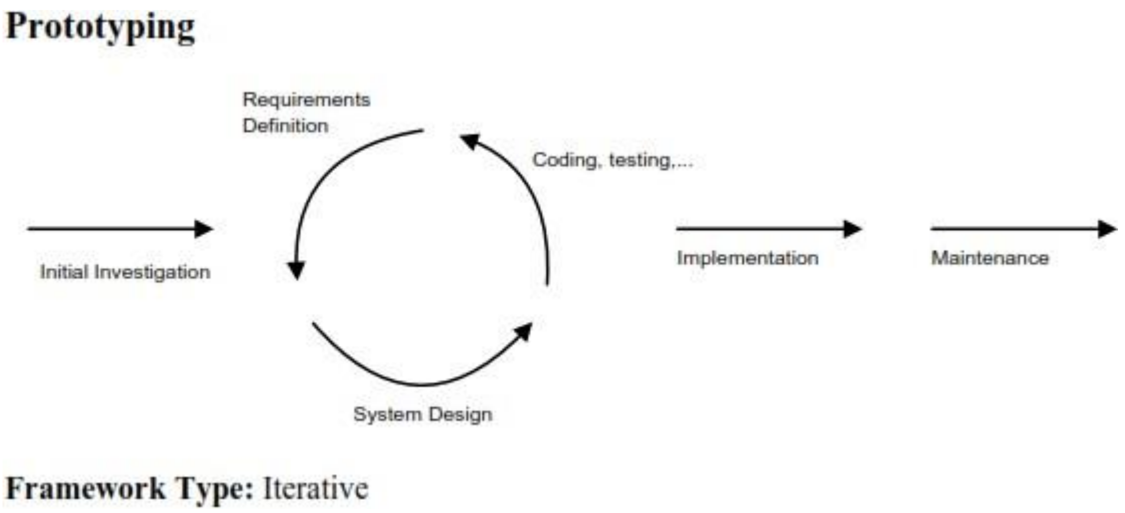
A model-driven and data-centered, but process-sensitive technique for planning, analyzing, and designing information systems. IE models are pictures that illustrate and synchronize the system's data and processes.

- The primary tool of IE is a data model diagram ie ERD

- **Prototyping**

  A small-scale, incomplete, but working sample of a desired system is developed.

  Has an Iterative process involving a close working relationship between the designer and the users.



Prototyping

Framework Type: Iterative

Key Benefits:

- Encourages and requires active end-user participation.
- Iteration accommodates end-users who tend to change their minds.
- Endorses philosophy that end-users won‘t know what they want until they see it.
- Active model that end-users can interact with.
- Errors can be detected earlier.
- Can increase creativity as it allows for quicker user feedback.
- Accelerates several phases of the life cycle.

Disadvantages and Pitfalls:

- Encourages ―code, implement, and repair‖ life cycle that cause maintenance nightmares.
- Still need systems analysis phases, but so easy to skip.
- Cannot completely substitute a prototype for a paper specification (like architect without a blueprint).
- Numerous design issues are not addressed by prototyping.
- Often leads to premature commitment to a design.

- Scope and complexity of the system can expand out of control.
- Can reduce creativity in designs.
- Often suffer from slower performance because of language considerations (rapidly becoming a non-issue).

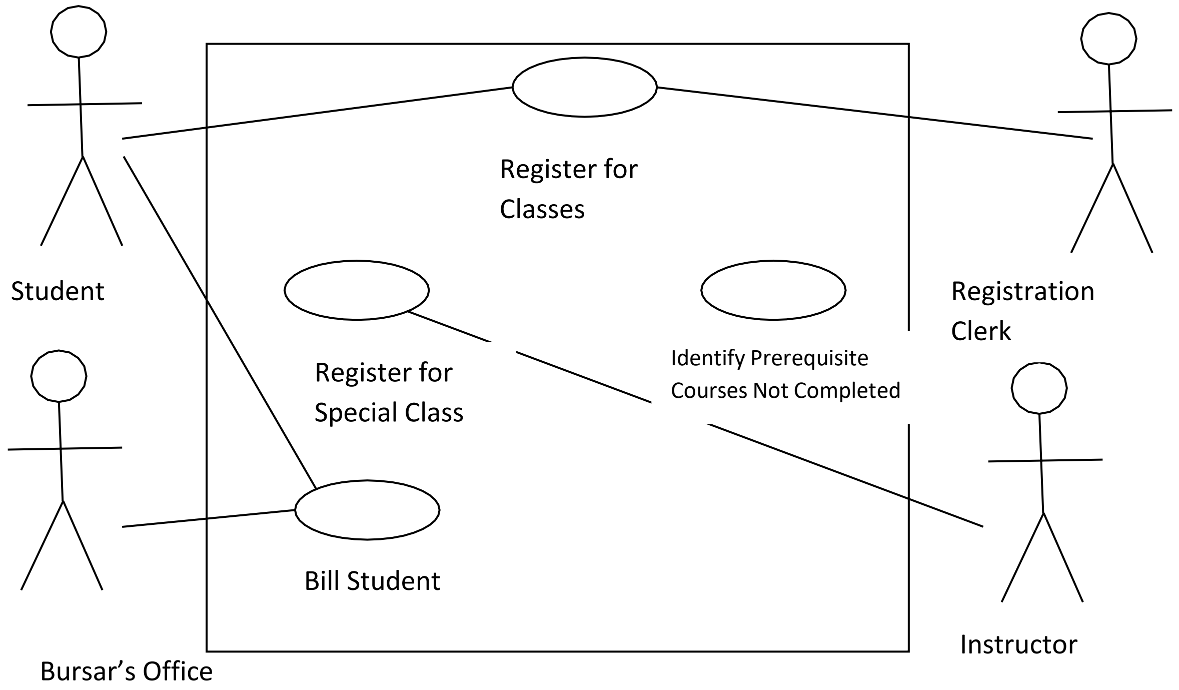
○ **Object-oriented Analysis**

Objected oriented analysis mainly uses USE cases and CASE diagrams. Use cases are a different way to model the business functionality of a business process that facilitates the development of information systems to support that process. Although common in object oriented systems analysis and design, use case modeling can also be used with more traditional methods for business process.

USE Case: A USE Case shows the behavior or functionality of a system. It consists of a set of possible sequences of interactions between a system and a user in a particular environment, possible sequences that are related to a particular goal. A use case describes the behavior of a system under various conditions as the system responds to request from principal actors

A principal actor initiates a request of the system, related to a goal, and the system responds. A use case can be stated as a present-tense verb phrase (what the system is supposed to do) and the object of the verb (what the system is to act on). For example, use case names would include Enter Sales Data, Compute Commission, Generate quarterly Reports

A use case model consists of **actors** and **use cases.** An actor is an external entity that interacts with the system. It is someone or something that exchanges information with the system.



Object oriented analysis and design (OOAD) is a development approach whose concern is to model the real world objects into a software. An object is a thing whose characteristics can be described. In the object oriented paradigm all living things and non living things are considered objects. And as such when a programmer is developing a s/w for an organization, he/she must consider all objects of interest to the organization and capture them in the software. Some of the objects may have real tangible existence such as employees‘ vehicles, chairs, tables etc. The goal of OOAD is to make systems elements more reusable, thus improving system quality and the productivity of systems analysis and design.

However some of the objects may have an abstract existence and therefore not visible or tangible, but the programmer must include them in the software as they are of interest to the organization.

Object oriented support the basic concept of a true OOP language namely:

- **Inheritance**: is the ability to acquire the characteristics, traits or functionality of an already existing objects of a given class. The class whose functionality are being inherited by another is said to be a super class or base class, or parent class. The class which acquires the functionality of another class is known as a sub class or a derived class, or child class. Through the process of inheritance, hierarchical tree structure can span out with different levels of hierarchy to give an impression of child, parent, grand parent etc.
- **Encapsulation:** it is the wrapping together of data and functions into a single unit of class. Data is not accessible to the outside world, only functions wrapped into the class can assess it. The insulation of data from operations is known as information hiding.
- **Polymorphism:** is the ability of an object to change or take a different form depending on the prevailing circumstances
- **Abstraction**: it is the hiding of the inner complex implementation details of an entity by providing an interface through which they can be manipulated or interacted with. Programmers implement abstraction through the creation of objects whose properties are concealed from the user but an interface is provided through the functions associated with that object.
- Structured

sub topic: Applying systems analysis tools

- **Flow charts**
  FLOWCHARTS.

❖ A **Flowchart** is a diagrammatic or pictorial representation of a program‘s algorithm.

❖ It is a chart that demonstrates the logical sequence of events that must be performed to solve a problem.

Types of Flowcharts.

There are 2 common types of Flowcharts:

1). System flowchart.

A *System flowchart* is a graphical model that illustrates each basic step of a data processing system.

It illustrates (in summary) the sequence of events in a system, showing the department or function responsible for each event.

2). Program flowchart.

This is a diagram that describes, in sequence, all the operations required to process data in a computer program.

A *program flowchart* graphically represents the types of instructions contained in a computer program as well as their sequence & logic.

PROGRAM FLOWCHARTS.

A Flowchart is constructed using a set of special shapes (or symbols) that have specific meaning. Symbols are used to represent operations, or data flow on a flowchart.

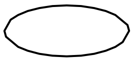
Each symbol contains information (short text) that describes what must be done at that point.

The symbols are joined by arrows to obtain a complete Flowchart. The arrows show the order in which the instruction must be executed.

SYMBOLS USED IN PROGRAM FLOWCHARTS.

Below is a standard set of symbols used to draw program flowcharts as created by ***American National Standard Institute*** (***ANSI***).

1. <u>Terminal symbol</u>.



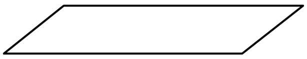
Ellipse (Oval in shape)

It is used to indicate the point atwhich a flowchart, a process or an algorithm begins & ends.

√ All Flowcharts must have a START & STOP symbol. The START/BEGIN symbol is the first symbol of a flowchart, & identifies the point at which the analysis of the flowchart should begin. The STOP/END symbol is the last symbol of a flowchart, & indicates the end of the flowchart.

√ The words **Begin**& **End** (or **Start**&**Stop**) should be inserted in the Terminal symbol.

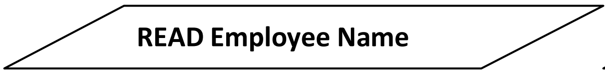
2. <u>Input or Output symbol</u>.



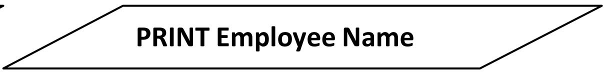
(*Parallelogram*)

- It is used to identify/specify an input operation or output operation.

For example;



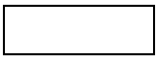
Input operation



Output operation

Note. The words mostly associated with I/O operations are **READ**&**PRINT**. READ describes the entry of computer data, while PRINT relates to the printed output of information.

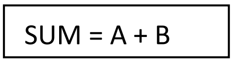
3. <u>Process symbol</u>.



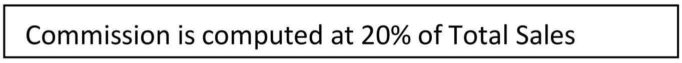
(*Rectangle*)

- **Process symbol** is used to indicate that a processing or data transformation is taking place.

The information placed within the process symbol may be an algebraic formula or a sentence to describe processing.

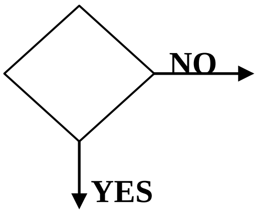


Processing defined as a Formula



Processing defined as a Sentence

4. <u>Decision symbol</u>.



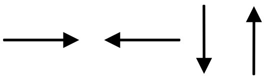
(*Rhombus*)

- It is used to indicate/ specify a condition or to show the decision to be made.
There are 2 main components of a Decision symbol:

(i). A question asked within the Decision symbol, that indicates the comparison / logical operation.

(ii). The results of the comparison (which are given in terms of **YES** or **NO**).
The arrows labeled YES or NO lead to the required action corresponding to the answer to the question.

5. **Flow lines.**

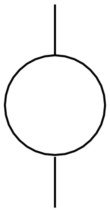


Flow lines with arrowheads are used to indicate the direction of processing of the program logic, i.e., they show the order in which the instructions are to be executed.

The normal flow of a flowchart is from *Top* to *Bottom*, and *Left* to *Right*.

Note. Flow lines should never cross each other.

6. **Connector symbol.**



Sometimes, a flowchart becomes too long to fit in a single page, such that the flow lines start crisscrossing at many places causing confusion & also making the flowchart difficult to understand.

The **Connector symbol** is used as a connecting point for arrows coming from different directions.

A Connector symbol is represented by a Circle, and a letter or digit is placed within the circle to indicate the link.

Note. Connectors do not represent any operation. They are used to connect two parts of a flowchart, indicating that the flow of data is not broken.

General guidelines for drawing a program flowchart.

1. A flowchart should have only one entry/starting point and one exit point (i.e., ensure that the flowchart has a logical start and finish).
2. The flowchart should be clear, neat and easy to follow.
3. Use the correct symbol at each stage in the flowchart.
4. The flowchart should not be open to more than one interpretation.
5. Avoid overlapping the lines used to show the flow of logic as this can create confusion in the flowchart.
6. Make comparison instructions simple, i.e., capable of YES/NO answers.

7. The logical flow should be clearly shown using arrows.
Note. A flowchart should flow from the Top to Bottom of a page, and from the Left to the Right.

8. Where necessary, use Connectors to reduce the number of flow lines.

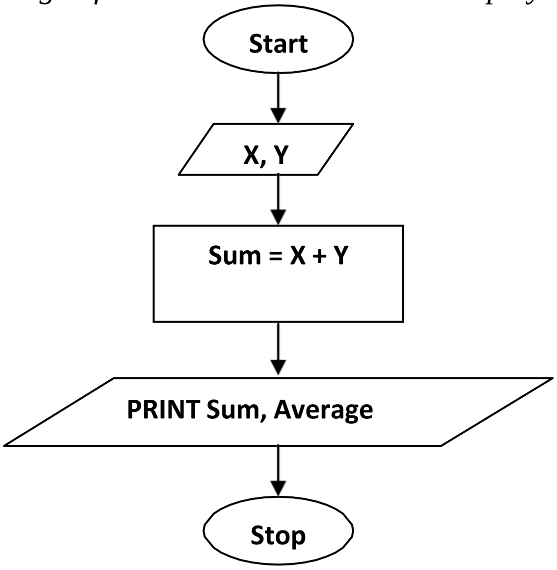
Connectors are helpful when a flowchart is several pages long, and where several loops are needed in the logic of the flowchart.

9. Check to ensure that the flowchart is logically correct & complete.

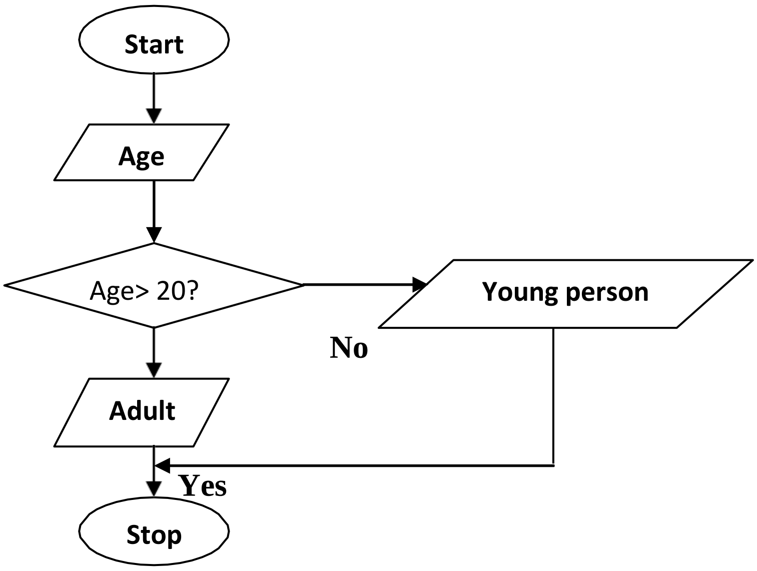
Example 1:

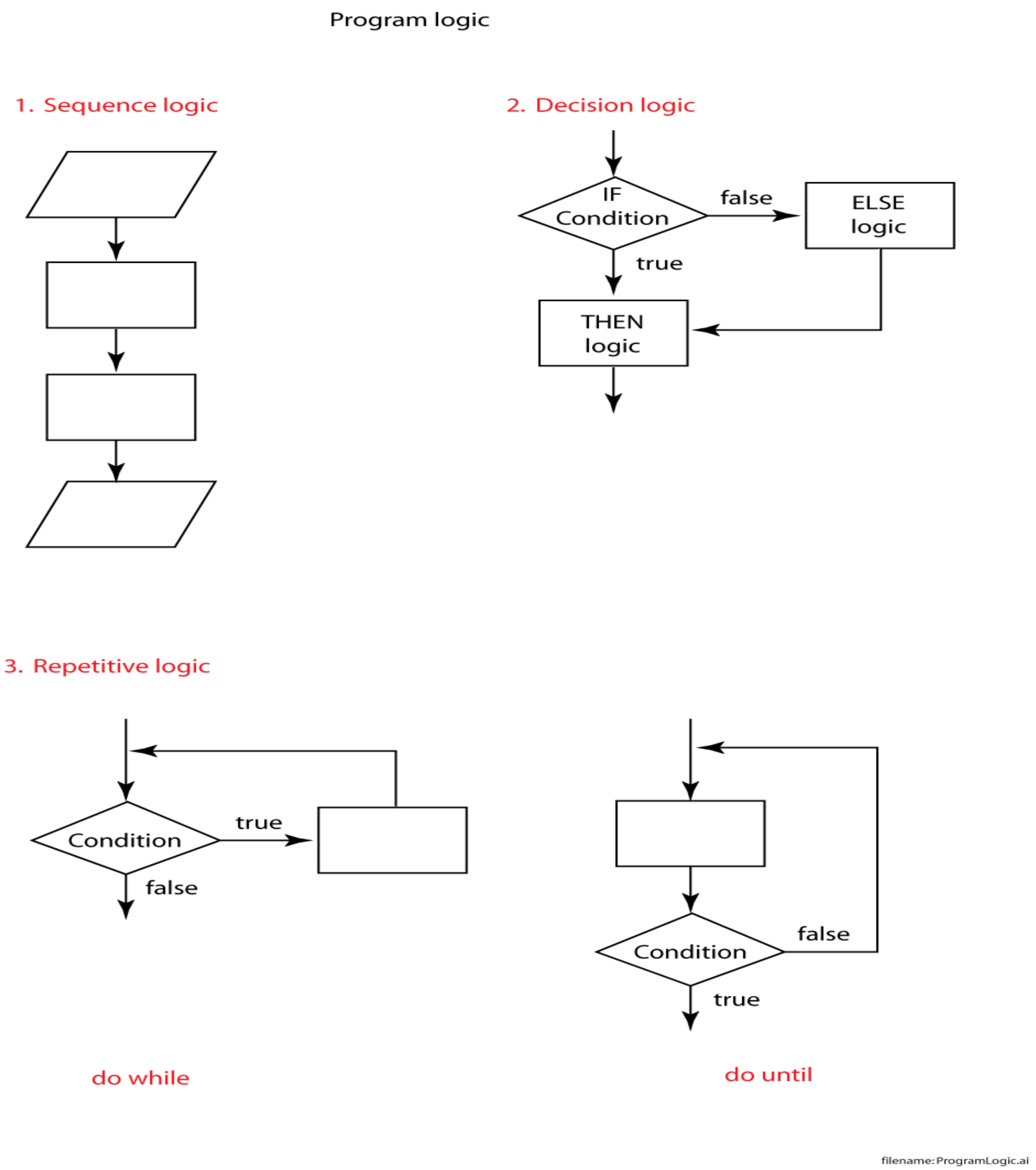
Draw a flowchart for a program that can be used to prompt the user to enter two numbers, find the sum and average of the two numbers and then display the output on the screen.

Example 2:



Design a flowchart for a program that can be used to classify people according to age. If a person is more than 20 years; output "Adult" else output "Young person".





Modules

Flowcharting symbols can be applied to manual processes – how a person completes a task – or to acomputerized function – as a precursor to writing the program. The analyst must often add information tomake a process comprehensible to humans or to machines. For example, the idea of gathering data or materialsbefore beginning a process may not be obvious at first. In this example, a person is calculating theaverage of student scores on a quiz. Imagine yourself doing the task. In this first iteration, the main functionsare identified:

1. Write down the score for each student‘s quiz
2. Add the score to the running total
3. Count the number of assignments
4. Divide the running total by the number of assignments

So far, so good. But a computer needs to be told much more:

```
1. Start the process of averaging student scores
2. Are the quizzes available?
        a. If not, get them
        b. If so, process
3. Running score = 0
4. Number of quizzes processed = 0;
5. while there are quizzes to process
        a. get the score from this quiz
        b. add the score to Running score
        c. add 1 to the number of quizzes processed
6. Divide the Running score by the Number of quizzes processed
7. Store the answer
8. End the process
```

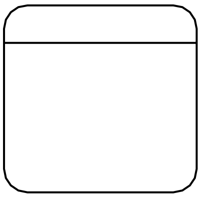
Since flowcharts are a graphic representation of the steps, the sequence is indicated by arrows. Note, though, that if there are many off-page connectors, the flowchart is probably too big. In this case,reconsider the modularization: can you identify redundant processes or processes whose level of granularityis inconsistent with others on the chart? In this situation, you may prefer a different charting technique.

- **DATA FLOW DIAGRAMS (DFDS)**

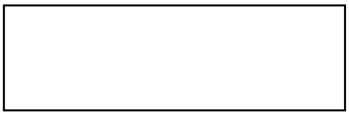
Data flow diagrams are used to represent the system being analysed. The diagrams/symbols used for basic elements are:

Process

This is any activity that alters data in any given way

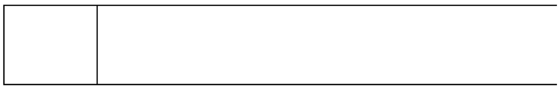


External Entity (source/sink)



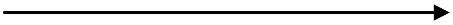
This is used to represent external entities or things which exist outside the system and which send or receive messages from the system. This may be users or other systems and they represent the source and destination(Sink) of the information flow that is being modelled.

Data Store.



This corresponds to master file of data whether they are manual, computerized or a database or any unit of stored data.

Data Flow



Shows movement of data between components of other systems.

To construct a DFD we use the top-down approach which starts with the most high level function and the major data flows in and out of this.

A diagram with small number of processes is constructed (Context diagram) to represent the top level. It is then expanded to produce a more detailed diagram (Level 0). The final stage is reached when each process corresponds to a single task or module which can clearly be simple and defined.

STEPS FOR DRAWING DFDs

Note: we are still at the analysis stage of the systems development. And so we are concerned with what we require of the system, how this will be achieved is left for the design stage.

Step 1:

Start by identifying all the source and destination entities of the system to be modelled. This is most likely a sub system of the total system which will have been partitioned into management parts.

The **nouns** and **noun phrases** in the given scenario description will give a guide to identify the **Entities** and **Data Stores.**

Step 2:

Draw the appropriate context or level 0 diagram.

Step 3:

Refine the context diagram by identifying all the processes within the main systems processes. The **VERBs** (***action doing words***) in the scenario description will help identify these **processes.**

Step 4:

Begin at source and decide which process is to receive that input, then decide where the output of that process is to go.

Repeat all these exercises for all the inputs and output processes. This will produce a full data flow chain in a level 1 dfd.

Step 5:

From step 4, take each process in turn and repeat step 4 for each, drawing a separate DFD for each process. These are the level 2 dfds.

NOTE: NEVER CONNECT A SOURCE DIRECTLY WITH A DESTINATION. IT MUST ALWAYS PASS THROUGH A PROCESS!!!!!!!

Data dictionary

A data dictionary describes the data items found in data flow diagrams and entity relationship diagrams. It provides a starting point for developing screen layouts, printed reports and programming logic. It provides an opportunity to detect design errors. It is maintained and expanded throughout the entire systems analysis and design process.

Example of Data Dictionary

completedEnrolmentForm = studentId
+ studentName
+ studentDateOfBirth
+ courseCode
+ courseEndDate

studentId = integer(6)
The first two digits represent the year of enrolment e.g. 07 for2007. The remaining four digits uniquely identify the student

studentNam = studentFirstName
+studentSurname

studentFirstName = char(15)
up to 15 characters are allowed

surname = char(15)
or family name. Up to 15 characters are allowed

studentDateOfBirth = short Date format e.g. 16May1989
year must be before 1989 i.e. they must be at least 16 years old on the
1st September when they enrol

courseCode = char(5)
up to five characters e.g. A0371 = Access toComputing course
Every courseCode has associated with it a courseTitle

courseEndDate= short Date format e.g. 30Jun2008
the date the person ceases to be a student

checkedEnrolmentForm = enrolmentForm
+ lecturerName
+ dateLecturerSigned

lecturerName = initials
+ surname

initials = char(3)
up to three characters allowed

CourseFile = { courseRecord }
CourseFile contains zero, one or many courseRecords

courseRecord = courseTitle
+ courseCode

courseTitle = char(25)
up to 25 characters e.g. BTEC Nat Dip Comp Studies

Entity Life History

A major reason for building computer-based information systems is to provide up-to-date and accurate information. Information is constantly changing or evolving, for example the number of beds available in a hospital, the price of petrol, and people's names and addresses. An information system must be able to keep track of these changes. Previous notes have described how the system is modelled from the viewpoint of information flows (DFDs) and from the viewpoint of the information (i.e. entities) that is held (ER modelling).

Entity Life Histories (ELH) model the system from the viewpoint of how the entities/information in a system evolves over time. What the ELHs show is the full set of changes that can possibly occur to the entities/information within the system, together with the context of each change.

Initially, each entity within a system is examined in isolation as this is a manageable unit of information to model. It is the stimuli of the changes that are modelled rather than the entity, a composite picture is formed, eventually specifying the full set of changes that will occur within the system.

An ELH is a diagrammatic representation of the life of a single entity, from its creation to its deletion. The life is expressed as the permitted sequence of events that can cause the entity to change. An event may be thought of as whatever brings a process into action to change entities, so although it is a process that changes the entity, it is the event that is the cause of the change.

Creation of Entity Life Histories

The necessary pre-requisites/ requirements for the development of life histories is knowledge of the following three ELH components:

- The system entities and their attributes.
- The events which affect one or more of the system entities during their lifetime in the system.
- A basic notation for describing graphically the chronological sequence in which the events and event sub structures may occur.

Structure of Entity Life Histories

The general form of an ELH follows certain conventions and has the appearance shown in the figure below.

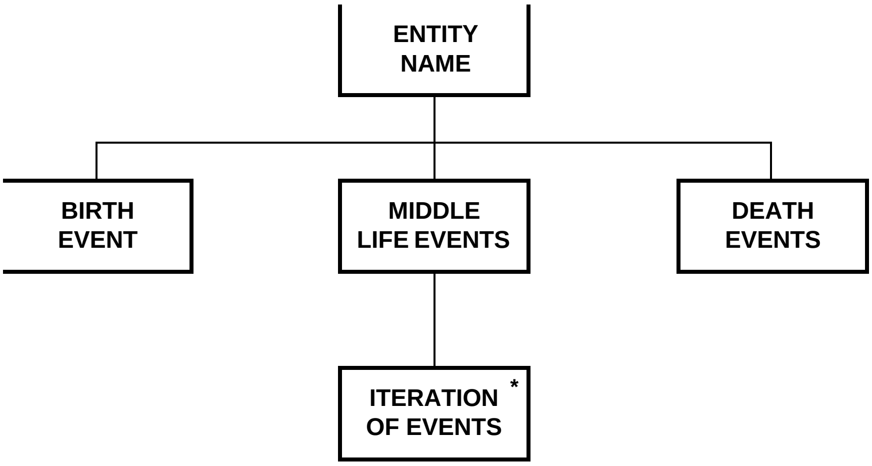


Figure One Structure of an ELH Diagram

The general structure of an ELH as shown in figure one shows that there exist three fundamental event types:

- Birth Events
- Death Events
- Middle Life (or update) Events

The corresponding effects of these are that they cause an occurrence of the entity to be:

- Created by the system.
- Deleted by the system.
- Modified in terms of changes to its attribute values.

Thus the initial questions to ask when attempting to identify life history events are:

- How does the system get to know about this entity?
- Why does it leave the system?
- What causes changes to its attributes?

All ELHs contain two types of node:

- Intermediate nodes and
- Leaf (or terminal) nodes.

In figure one Birth event, Iteration of events and Death event are all leaf nodes, Middle life events is an intermediate node. The root of the tree gives the name of the entity whose life we wish to analyse. An ELH diagram is constructed for each entity in the ER diagram.

The passage of time is assumed to be a uniform flow from left to right of the diagram. At time zero in the life of an entity the system gets to know about it by virtue of the arrival of one (or generally one of a set) of system events resulting in the creation of an occurrence of the entity in the system. The middle life, the period between birth and death, is typically a set of recurring or iterating events causing changes to the entity. This period represents the majority of the life of the entity in the system, as such this part of the diagram can get quite complex. Eventually, the life is terminated by the arrival of a 'death' event causing that entity occurrence to be deleted and removed from the system.

ELH Notation

The basic notation of life histories is used after event identification to record graphically the sequence in which events and event sub-structures may legally occur. The number of distinct symbols for initial ELH work is limited to only three, which together with the convention for representing the passage of time, complete the notation. The symbols are:

□	Entity names, group headings and events.
*****	Event repetition or iteration.
0	A selection between two or more events.

Control Structures

There are three control structures that can be applied to an ELH:

1. Event Sequence
2. Event Selection
3. Event Iteration

2.2.1 Event Sequence

A sequence is represented by a series of boxes reading from left to right as shown in figure two.

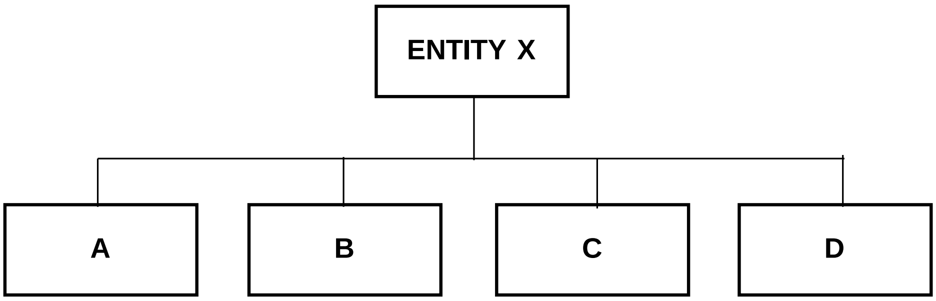


Figure Two Event Sequence

The box labelled A will always be the first to occur, followed by B which in turn is followed by C then D. **This is the only possible sequence**. Although the sequence may be thought of as a progression through time, there is no indication of the time intervals between the boxes within a sequence. These could span minutes, hours, days, or years.

Event Selection

A selection defines a number of effects or nodes that are alternatives to one another at a particular point in the ELH. Note that one, and only one, of two or more possible events will occur at a particular time in a sequence. A selection is represented by a set of boxes with circles in the top right corners as shown in figure three.

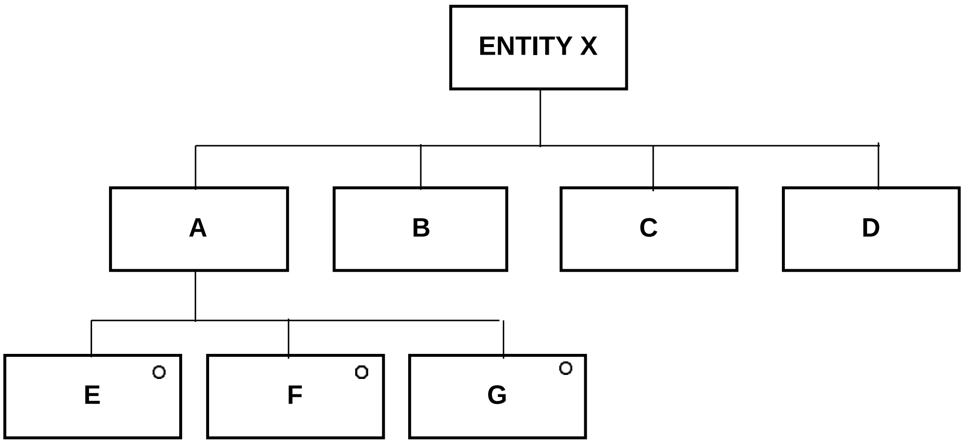


Figure Three Event Selection

As node A is at the beginning of the ELH, this diagram shows that an occurrence of entity X must be created by only one of three events: E, F or G.

Event Iteration

An iteration is where an effect or node may be repeated any number of times at the same point within an ELH. A restriction upon the iteration is that each occurrence of the iteration must be complete before the next begins. This is most relevant where a node is being repeated. An iteration is represented by an asterisk in the top right-hand corner of a box as shown in figure four.

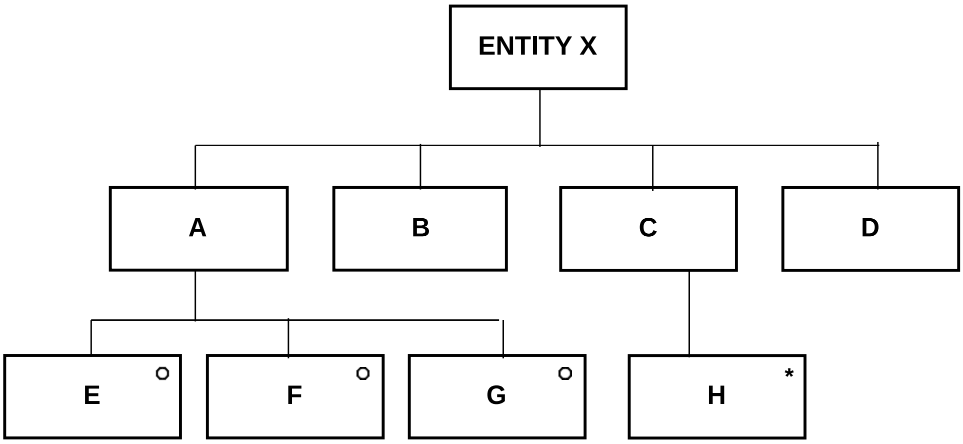


Figure Four Event Iteration

After entity X has been created by E, F or G, the event H may affect the entity any number of times. Here it is important to note that 'any number of times' includes none, so an iteration is another way of showing that something may or may not occur.

The 'Levels' in an ELH

An ELH structure can extend to several levels of detail where each level is represented by a horizontal bar, immediately above which will be a group heading and below it the 'details' which collectively describe the construct of the heading.

Life history structures should be minimal structures in terms of levels necessary to describe an entity's lifetime as a collection of system events. A fundamental requirement that directly affects the number of levels shown in particular situations is that since all the boxes below a horizontal line describe the construct of the box above it, the mixing of constructs in boxes below a horizontal line would result in a contradictory description. In practice this means that all boxes hanging below a horizontal line must have the same construct (sequence of event types, iterations or a selection). Whenever this requirement cannot be met, new levels and group headings must be introduced to rectify the situation.

3 Worked Example (Ashworth & Goodland, 1990)

Imagine that Maurice has decided to open a bank account at Cash & Grabbs Bank. When Maurice has persuaded Mr Cash, the manager, that he would be suitable as a customer, Mr Cash turns to his computer terminal and records Maurice's new bank account code in the system. The ER diagram of the bank computer system contains an entity called Bank Account. The event occurrence that creates the Maurice occurrence of the entity Bank Account is the opening of the account by Mr Cash.

This event occurrence and the ones that follow it are:

- Account opened for Maurice.
- Cash Deposit £2000
- Cheque Cashed for £20
- Direct Deposit £1000
- Cheque Cashed for £20
- Direct Deposit £2000 etc.

Jonnes is another customer at the bank, the event occurrences that affect his bank account are:

- Account Opened for Jonnes
- Pay Deposit by Credit Transfer £500
- Cheque Cashed for £200

- Cheque Cashed for £300
- Cheque Cashed for £300

Other accounts may behave in similar ways, none of which are precisely the same. However, it is possible to build up a general picture that will fit all occurrences of Bank Accounts at the Cash & Grabbs Bank. Basically, all accounts are opened, and several deposits and several withdrawals may be made. The way that these events affect the entity Bank Account is shown in figure five.

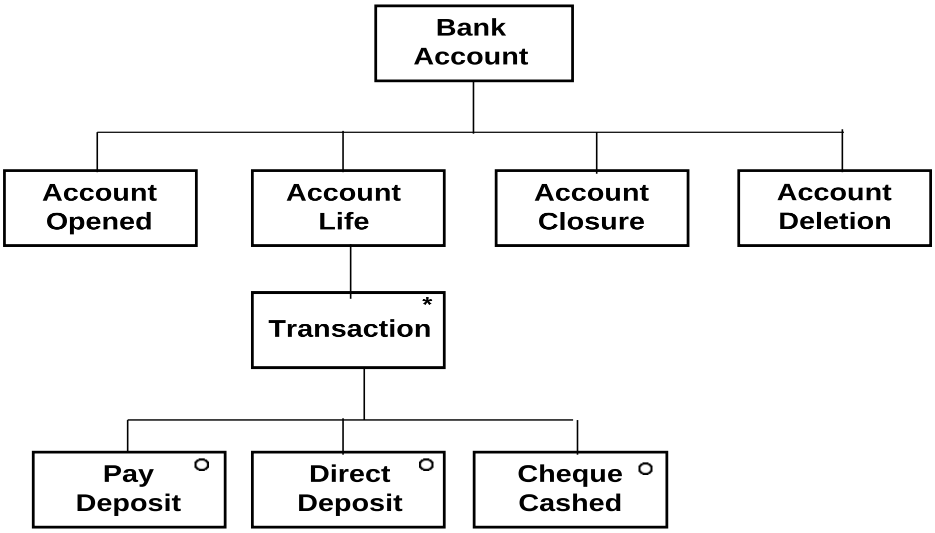


Figure Five ELH for Bank Account (Cash & Grabb Bank)

This ELH shows that the first event to affect the entity Bank Account will be Account Opened for all occurrences. Next, the account has a life which is a series of transactions. Each Transaction is one of: a Pay Deposit, a Direct Deposit, or a Cheque Cashed. After an undefined number of Transactions have taken place, the Account will be closed and finally deleted.

TOPIC 7: SYSTEMS DESIGN AND DEVELOPMENT

THEORY

Meaning and importance of system design

Systems design is the process of defining the architecture, components, modules, interfaces, and data for a **system** to satisfy specified requirements. **Systems design** could be seen as the application of **systems** theory to product development.

System analysis describes what a system should do to meet the information needs of users. System design specifies how the system will accomplish this objective. Systems design consists of design activities, which produce systems specifications satisfying the functional requirements developed in the systems analysis stage. These specifications are used as the basis for:

1. Software development
2. Hardware acquisition
3. System testing
4. Other activities of the implementation stage.

Qualities of a good System Design

Quality attributes. A good systems design should have the following:

Quality attribute	**Definition**
Agility	The ability of a system to both be flexible and undergo change rapidly.
Flexibility	The ease with which a system or component can be modified for use in applications or environments, other than those for which it was specifically designed.
Interoperability	The ability of two or more systems or components to exchange information and use the information that has been exchanged.
Maintainability	The aptitude of a system to undergo repair and evolution. (1) The ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment. (2) The ease with which a hardware system or component can be retained in, or restored to, a state in which it can perform its required functions.
Performance	The responsiveness of the system—that is, the time required to respond to events or the number of events processed in some interval of time. Performance qualities are often expressed by the number of transactions per unit time, or by the amount of time that it takes to complete a transaction with the system.
Reliability	The ability of the system to keep operating over time. Reliability is usually measured by mean time to failure.

	The degree to which a software module or other work product can be used in more than one computing program or software system. This is typically in the form of reusing software that is an encapsulated unit of functionality.
Scalability	The ability to maintain or improve performance while system demand increases.
Security	A measure of the system's ability to resist unauthorized attempts at usage and denial of service, while still providing its services to legitimate users. Security is categorized in terms of the types of threats that might be made to the system.
Supportability	The ease with which a software system can be operationallymaintained.
Testability	The degree to which a system or component facilitates the establishment of test criteria and the performance of tests to determine whether those criteria have been met.
Usability	The measure of a user's ability to utilize a system effectively The ease with which a user can learn to operate, prepare inputs for, and interpret outputs of a system or component. A measure of how well users can take advantage of some system functionality. Usability is different from utility, which is a measure of whether that functionality does what is needed.

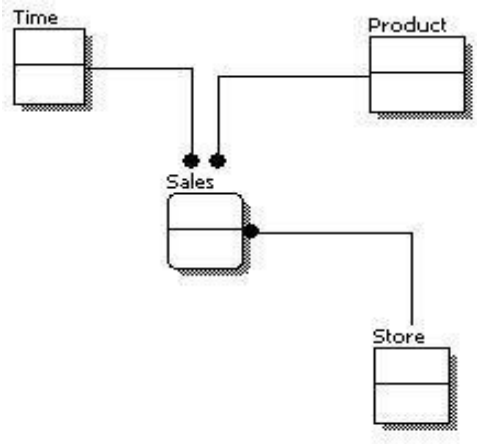
System Design Models

A conceptual data model identifies the highest-level relationships between the different entities. Features of conceptual data model include:

- Includes the important entities and the relationships among them.
- No attribute is specified.
- No primary key is specified.

The figure below is an example of a conceptual data model.

Conceptual Data Model



From the figure above, we can see that the only information shown via the conceptual data model is the entities that describe the data and the relationships between those entities. No other information is shown through the conceptual data model.

LOGICAL MODEL

A logical data model describes the data in as much detail as possible, without regard to how they will be physical implemented in the database. Features of a logical data model include:

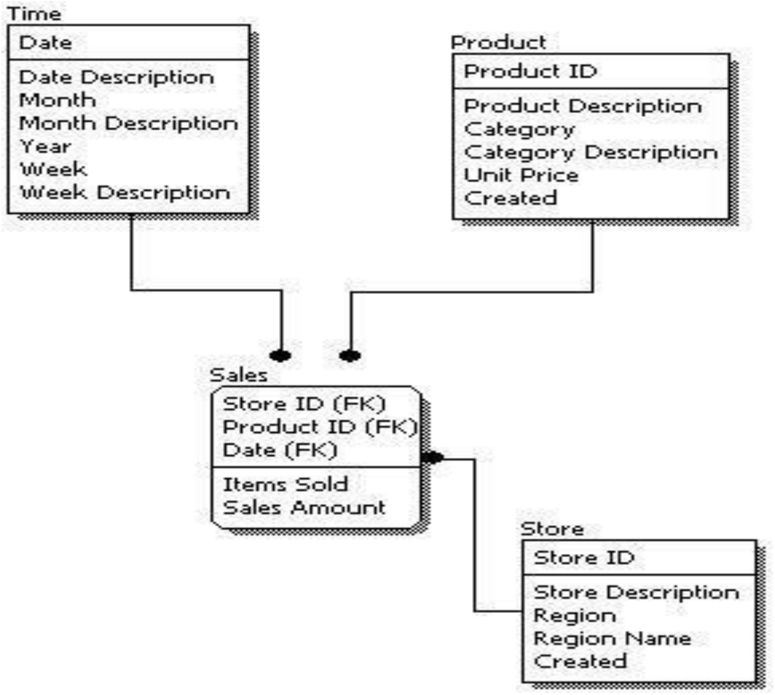
- Includes all entities and relationships among them.
- All attributes for each entity are specified.
- The primary key for each entity is specified.
- Foreign keys (keys identifying the relationship between different entities) are specified.
- Normalization occurs at this level.

The steps for designing the logical data model are as follows:

1. Specify primary keys for all entities.
2. Find the relationships between different entities.
3. Find all attributes for each entity.
4. Resolve many-to-many relationships.
5. Normalization.

The figure below is an example of a logical data model.

Logical Data Model



Comparing the logical data model shown above with the conceptual data model diagram, we see the main differences between the two:

- In a logical data model, primary keys are present, whereas in a conceptual data model, no primary key is present.
- In a logical data model, all attributes are specified within an entity. No attributes are specified in a conceptual data model.
- Relationships between entities are specified using primary keys and foreign keys in a logical data model. In a conceptual data model, the relationships are simply stated, not specified, so we simply know that two entities are related, but we do not specify what attributes are used for this relationship.

Physical data model represents how the model will be built in the database.

Logical Design-Summary

A logical design is a conceptual, abstract design. You do not deal with the physical implementation details yet; you deal only with defining the types of information that you need.

The process of logical design involves arranging data into a series of logical relationships called entities and attributes. An *entity* represents a chunk of information. In relational databases, an entity often maps to a table. An *attribute* is a component of an entity and helps define the uniqueness of the entity. In relational databases, an attribute maps to a column.

Physical Data Model

A physical database model shows all table structures, including column name, column data type, column constraints, primary key, foreign key, and relationships between tables. Features of a physical data model include:

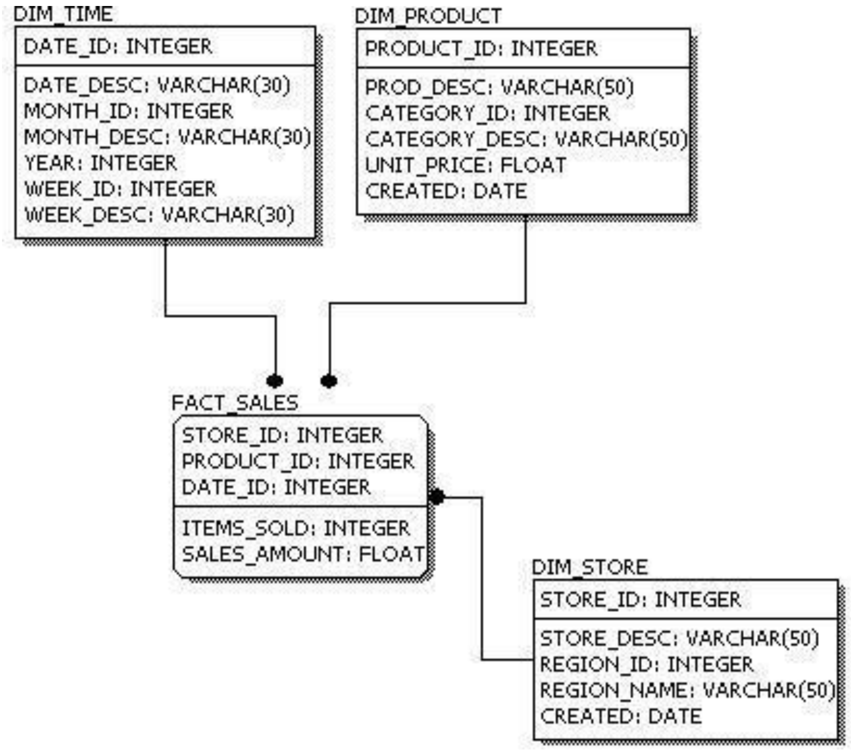
- Specification all tables and columns.
- Foreign keys are used to identify relationships between tables.
- De-normalization may occur based on user requirements.
- Physical considerations may cause the physical data model to be quite different from the logical data model.
- Physical data model will be different for different RDBMS. For example, data type for a column may be different between MySQL and SQL Server.

The steps for physical data model design are as follows:

1. Convert entities into tables.
2. Convert relationships into foreign keys.
3. Convert attributes into columns.
4. Modify the physical data model based on physical constraints / requirements.

The figure below is an example of a physical data model.

Physical Data Model



Comparing the physical data model shown above with the logical data model diagram, we see the main differences between the two:

- Entity names are now table names.
- Attributes are now column names.
- Data type for each column is specified. Data types can be different depending on the actual database being used.

Summary of Logical vs. Physical

At this moment in time, the business requirements are already defined, the scope of application has been agreed upon, and you have a conceptual design. So now you need to translate your requirements into a system deliverable. In this step, you create the logical and physical design for the data warehouse and, in the process, define the specific data content, relationships within and between groups of data, the system environment supporting your data warehouse, the data transformations required, and the frequency with which data is refreshed.

In the *logicaldesign***,** you look at the logical relationships among the objects.

In the *physicaldesign*, you look at the most effective way of storing and retrieving the objects.

Your design should be oriented toward the needs of the end users. End users typically want to perform analysis and look at aggregated/overall data, rather than at individual transactions. Your design is driven primarily by end-user utility, but the end users may not know what they need until they see it. A well-planned design allows for growth and changes as the needs of users change and evolve.

Beginning with the logical design, you mainly focus on the information requirements without getting bogged down immediately with implementation detail.

System design components

- **Input**

Users deserve quality output. The quality of system input determines the quality of system output. It is vital that input forms, displays, and interactive Web documents be designed with this critical relationship in mind. Well-designed input forms, displays, and interactive Web fill-in forms should meet the objectives of effectiveness, accuracy, ease of use, consistency, simplicity, and attractiveness. All these objectives are attainable through the use of basic design principles, the knowledge of what is needed as input for the system, and an understanding of how users respond to different elements of forms and displays.

Effectiveness means that input forms, input displays, and fill-in forms on the Web all serve specific purposes for users of the information system, whereas accuracy refers to design that ensures proper completion. Ease of use means that forms and displays are straightforward and require no extra time for users to decipher. Consistency means that all input forms, whether they are input displays or fill-in forms on the Web, group data similarly from one application to the next, whereas simplicity refers to keeping those same designs uncluttered in a manner that focuses the user‘s attention. Attractiveness implies that users will enjoy using input forms because of their appealing design

Process Design

The process design activity focuses on the design of software resources, that is, computer programs and of procedures needed by the proposed information system. It concentrates on developing detailed specifications for the program modules that will have to be purchased as software packages or developed by custom programming. Process design produces:

1. Detailed program specifications and procedures needed to meet the user interface and data design specifications that are developed.
2. Produces specifications that meet the functional control and performance requirements developed in the analysis stage.

- **Reports**

Primarily used to convey large portions of data in the database. Output can be specially formatted for a variety of purposes such as printing mailing labels.

- **Code design**

The system design needs to be implemented to make it a workable system. This demands the coding of design into computer understandable language, i.e., programming language. This is also called the programming phase in which the programmer converts the program specifications into computer instructions, which we refer to as programs. It is an important stage where the defined procedures are transformed into control specifications by the help of a computer language. The programs coordinate the data movements and control the entire process in a system. It is generally felt that the programs must be modular in nature. This helps in fast development, maintenance and future changes, if required.

- **Database design**

If you are making a database then you need to include an E-R Diagram. You should show the diagram and explain what each of the relationships mean. For example:

- A student can have many bookings
- Each booking only has one student
- Each booking only has one book

- A book can be involved in many bookings

Sample of planned SQL queries

If you are making a database you need to write about the SQL that you use to **SELECT**, **INSERT**, **UPDATE** and **DELETE** things from your tables. In some cases you might also be writing the code behind making the individual tables, the data definition language. In fact those headings are exactly what you need to use.

Reserved words

When you are querying your database you might get some unexpected errors, where a query fails to run when it doesn't appear to have any problem with it. This may be due to using a reserved word in your query. SQL has a lot of reserved words, words that have special meanings, and if you use one of these in a query it **won't** treat it as a field name. For example:

```
SELECT Username, Password FROM tblUsers
```

This might bring up an error as Password is a reserved word in SQL. To get by this problem you might want to change your fieldnames to something a little more sensible or put the fieldname in square brackets:

```
SELECT Username,[Password]FROM tblUsers
```

There are many other reserved words out there, so be careful:

PERCENT, PLAN,PRECISION,EXISTS,PRIMARY, PRINT, PUBLIC,
BACKUP,FOREIGN,READ,FREETEXT,FROM,REFERENCES, BULK,
FULL, RESTORE,GROUP,IDENTITY, RULE, SAVE,INDEX,SELECT,
STATISTICS,KEY,TABLE,NATIONAL,DATABASE,UNION,DELETE,
DISK,ON,USER, PASSWORD

Different databases have different sets of reserved words.

Note: If you are not using a SQL server (for example, using MySQL with PHP) you may need to use `backticks` instead of square bracket notation.

SELECT

This section should list all the SQL you have written that returns selections of records. You should describe in plain English where the SQL is used and what it does and then include the SQL with any annotations that you need. For example:

English Description:
This SQL statement selects the details (ID, Name, Price, Description) of a product which has been

selected by the user. So that the user can view the product detail on the preview screen before buying the product

SQL:

```
SELECT ID, Name, Price, Description FROM Products
WHERE ID=?
```

INSERT

This section should list all the SQL you have written that inserts new records into your database. You should describe in plain English where the SQL is used and what it does and then include the SQL with any annotations that you need. For example:

English Description:
This SQL statement adds a new high score along with the date the score was achieved for a users account.

SQL:

```
INSERTINTO Scores (ID, DateofScore, Score)
VALUES(?,?,?)
```

UPDATE

This section should list all the SQL you have written that updates an old record with new details. You should describe in plain English where the SQL is used and what it does and then include the SQL with any annotations that you need. For example:

English Description:
This SQL statement updates the date of birth of a user, after they have updated the form that launches when they first log into the program.

SQL:

```
UPDATEUSER
SET DoB=?
WHERE ID=?
```

Data Definition Language

If you have designed your database tables using a program such as Open ModelSphere you can then define the code behind making your tables which you can then feed into MySQL or MSSQL to create your database. Get some credit for this and list your DDL. For example, the command to create a table named **employees** with a few sample columns would be:

```
CREATETABLE employees (
   id           INTEGERPRIMARYKEY,
   first_name   CHAR(50)NULL,
   last_name    CHAR(75)NOTNULL,
   dateofbirth  DATENULL
);
```

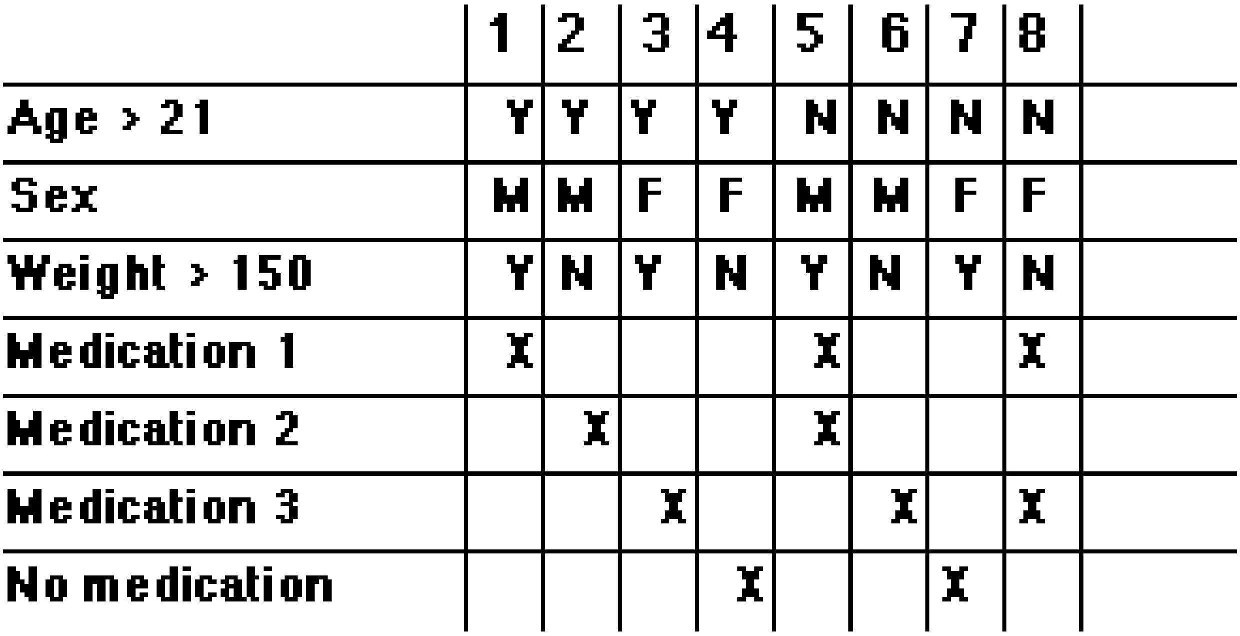
- **File design**

System design tools

- **Decision tables**

A decision table is created by listing all the relevant variables (sometimes known as *conditions* or *inputs*) and all the relevant actions on the left side of the table; note that the variables and actions have been conveniently separated by a heavy horizontal line. In this example, each variable is a logical variable, meaning that it can take on the value of true or false.

In many applications, it is easy (and preferable) to express the variables as binary (true-false) variables, but decision tables can also be built from multivalued variables; for example, one could build a decision table with a variable called –customer-age‖ whose relevant values are –less than 10,‖ –between 10 and 30,‖ and –greater than 30.‖



	1	2	3	4	5	6	7	8
Age > 21	Y	Y	Y	Y	N	N	N	N
Sex	M	M	F	F	M	M	F	F
Weight > 150	Y	N	Y	N	Y	N	Y	N
Medication 1	X				X			X
Medication 2		X			X			
Medication 3			X			X		X
No medication				X			X	

Next, every possible combination of values of the variables is listed in a separate column; each column is typically called a *rule*. A rule describes the action (or actions) that should be carried out for a specific combination of values of the variables. At least one action needs to be specified for each rule (i.e., for each vertical column in the decision table), or the behavior of the system for that situation will be unspecified.

If there are N variables with binary (true-false) values, then there will be 2^N distinct rules; thus, if there are 3 conditions, there will be 8 rules.

You must discuss *each* rule with the user to ensure that you have identified the correct action, or actions, for each combination of variables. It is quite common, when doing this, to find that the user has never thought about certain combinations of variables or that they have never occurred in his or her experience.

Advantages of DT

1. The advantage of the decision table approach is that you can concentrate on one rule at a time.

2. It does not imply any particular form of implementation. That is, when the systems analyst delivers the decision table (along with the DFDs, etc.) to the designer/programmer, there is a tremendous freedom of choice in terms of implementation strategy: the decision table can be programmed with nested **IF** statements, with a **CASE** construct or a **GO TO DEPENDING ON** construct in COBOL; in the extreme case, a decision table code generator can *automatically* generate code from the decision table. Thus, decision tables are often referred to as a *nonprocedural* system modeling tool, for they do not require any specific procedural algorithm to carry out the required actions.

To summarize, we must go through the following steps to create a decision table for a process specification:

1. Identify all the conditions, or variables, in the specification. Identify all the values that each variable can take on.
2. Calculate the number of combinations of conditions. If all the conditions are binary, then there are 2N combinations of N variables.
3. Identify each possible action that is called for in the specification.
4. Create an ―empty‖ decision table by listing all the conditions and actions along the left side and numbering the combinations of conditions along the top of the table.
5. List all the combinations of conditions, one for each vertical column in the table.
6. Examine each vertical column (known as a rule) and identify the appropriate action(s) to be taken.
7. Identify any omissions, contradictions, or ambiguities in the specification (e.g., rules in the decision table for which the specification does not indicate that actions should be taken).
8. Discuss the omissions, contradictions, and ambiguities with the user.

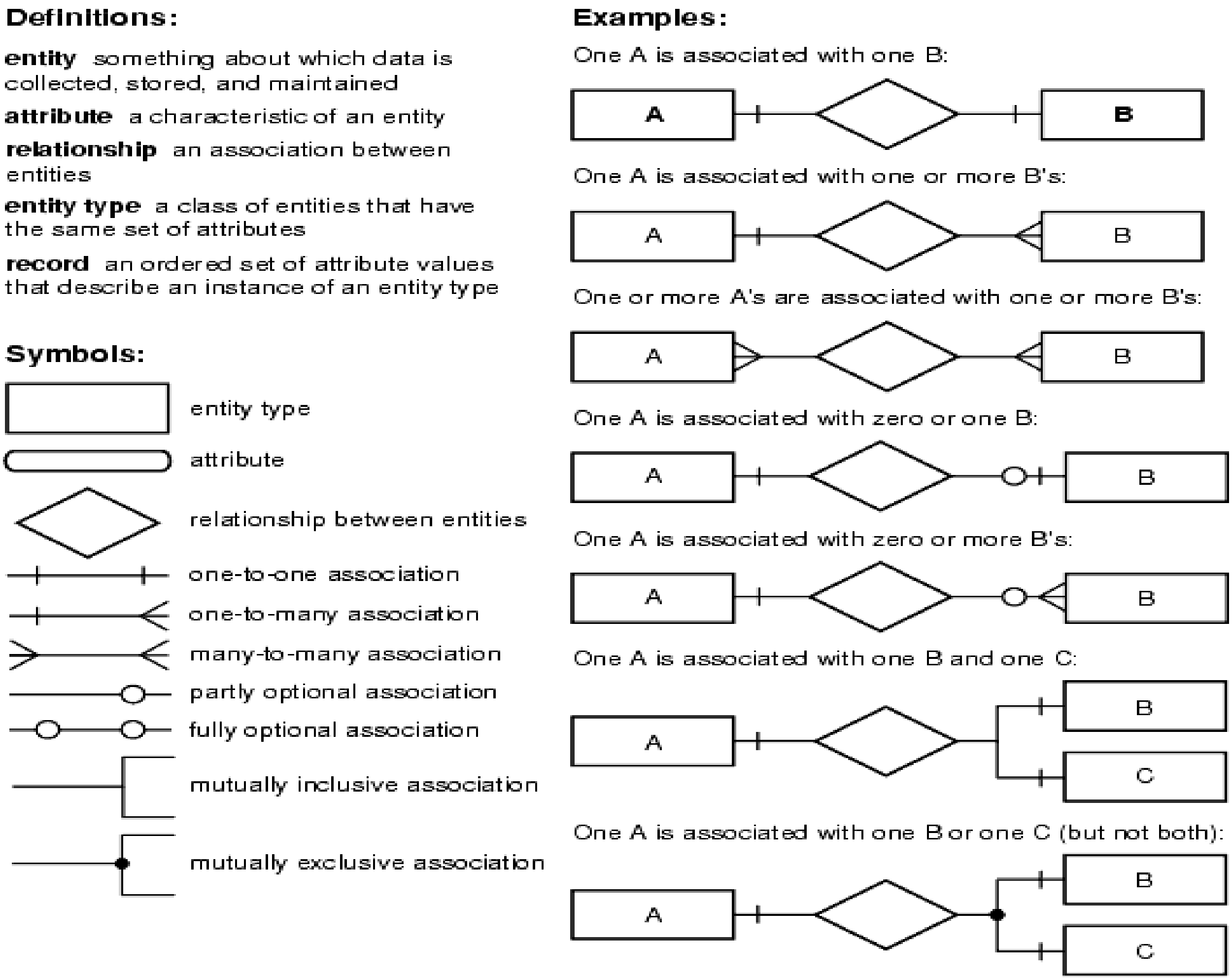
- **Structured English**

```
Decision Structure
    Only IF a condition is true
    Complete the following
    Statements; otherwise, jump to the next statement
    ELSE
Example  code
IF code A is true
    THEN implement action A
    ELSE implement Action B
ENDIF
```

ENTITY RELATIONSHIP DIAGRAMS (ERDs)

An ERD is a model that identifies the concepts or entities that exist in a system and the relationships between those entities. An ERD is often used as a way to visualize a relational database: each entity represents a database table, and the relationship lines represent the keys in one table that point to specific records in related tables. ERDs may also be more abstract, not necessarily capturing every table needed within a database, but serving to diagram the major concepts and relationships.

Rules for Drawing ERDs



Definitions:

entity something about which data is collected, stored, and maintained

attribute a characteristic of an entity

relationship an association between entities

entity type a class of entities that have the same set of attributes

record an ordered set of attribute values that describe an instance of an entity type

Symbols:

entity type
attribute
relationship between entities
one-to-one association
one-to-many association
many-to-many association
partly optional association
fully optional association
mutually inclusive association
mutually exclusive association

Examples:

One A is associated with one B:

One A is associated with one or more B's:

One or more A's are associated with one or more B's:

One A is associated with zero or one B:

One A is associated with zero or more B's:

One A is associated with one B and one C:

One A is associated with one B or one C (but not both):

Terms used in ERDs

Entities

Entities are concepts within the data model. Each entity is represented by a box within the ERD. Entities are abstract concepts, each representing one or more instances of the concept in question. An entity might be considered a container that holds all of the instances of a particular thing in a system. Entities are equivalent to database tables in a relational database, with each row of the table representing an instance of that entity.

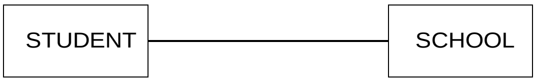
Remember that each entity represents a container for instances of the thing in question. The diagram below has an entity for ‒student‖ and another for ‒school.‖ This indicates that the system being modelled may contain one or more students and one or more schools.

STUDENT
SCHOOL

So far, no relationship between students and schools has been indicated.

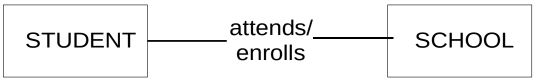
Relationships

Relationships are represented by lines between entities. Relationship lines indicate that each instance of an entity may have a relationship with instances of the connected entity, and vice versa.



The diagram above now indicates that students may have some relationship with schools. More specifically, there may be a relationship between a particular student (an instance of the student entity) and a particular school (an instance of the school entity).

If necessary, a relationship line may be labelled to define the relationship. In this case, one can infer that a student may attend a school, or that a school may enrol students. But if necessary, this relationship could be labelled for clarification:



Read the first relationship definition, ―attends,‖ when tracing the relationship left to right or top to bottom. Read the second definition, ―enrols,‖ when tracing the relationship right to left or bottom to top.

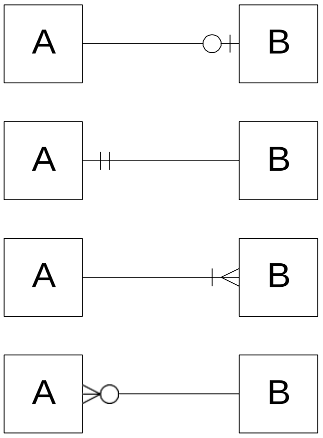
Optionality and Cardinality

Symbols at the ends of the relationship lines indicate the optionality and the cardinality of each relationship. ―Optionality‖ expresses whether the relationship is optional or mandatory. ―Cardinality‖ expresses the maximum number of relationships.

As a relationship line is followed from an entity to another, near the related entity two symbols will appear. The first of those is the optionality indicator. A circle (○) indicates that the relationship is optional—the minimum number of relationships between each instance of the first entity and instances of the related entity is zero. One can think of the circle as a zero, or a letter O for ―optional.‖ A stroke (|) indicates that the relationship is mandatory—the minimum number of relationships between each instance of the first entity and instances of the related entity is one.

The second symbol indicates cardinality. A stroke (|) indicates that the maximum number of relationships is one. A ―crows-foot‖ (<) indicates that many such relationships between instances of the related entities might exist.

The following diagram indicates all of the possible combinations:



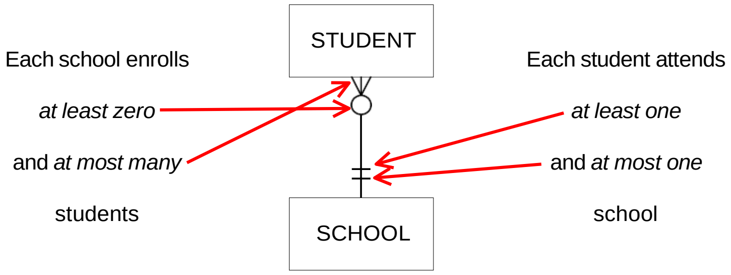
Each instance of A is related to a minimum of zero and a maximum of one instance of B

Each instance of B is related to a minimum of one and a maximum of one instance of A

Each instance of A is related to a minimum of one and a maximum of many instances of B

Each instance of B is related to a minimum of zero and a maximum of many instances of A

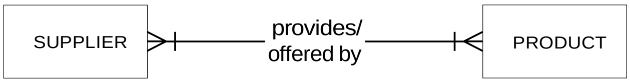
In our model, we wish to indicate that each school may enrolmany students, or may not enrol any students at all. We also wish to indicate that each student attends exactly one school. The following diagram indicates this optionality and cardinality:



It is important to note that relationship **optionality** and **cardinality** constraints apply specifically to the system being modeled, not to all possible systems. According to the example modelled above, a school might not enrol any students—that relationship is optional. A school without students is not much of a school, and indeed if the system being modeled were a school system enrollment database, the relationship would probably be mandatory. However, if the system being modeled is an extracurricular honors program, there may be schools that have no students currently participating in the program.

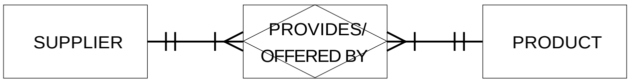
Bridge Entities

When an instance of an entity may be related to multiple instances of another entity and vice versa, that is called a ―many-to-many relationship.‖ In the example below, a supplier may provide many different products, and each type of product may be offered by many suppliers:



While this relationship model is perfectly valid, it cannot be translated directly into a relational database design. In a relational database, relationships are expressed by keys in a table column that point to the correct instance in the related table. A many-to-many relationship does not allow this relationship expression, because each record in each table might have to point to multiple records in the other table.

In order to build a relational database that captures this relationship, it is necessary to build a bridge between the two entities that uniquely expresses each relationship instance. This can be modeled in an ERD with a ―bridge entity,‖ an entity box containing a diamond, which may replace the many-to-many relationship. (The diamond is used in other ER modelling systems to indicate relationships, or may be viewed as the joining—the bridge—of the many-to-many ―crows-feet‖).

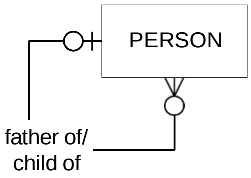


This diagram expresses the same relationship as the diagram above. Each instance of the ―provides‖ bridge entity indicates that a certain supplier can provide a certain product.

In addition to explicitly depicting a relational database structure that can capture a many-to-many relationship, the bridge entity has an additional function in abstract entity-relationship modelling: A bridge entity may capture attributes that are specific to the relationship between instances of the bridged entities. In the supplier and product example, a product does not have an inherent cost; it only has a cost in relation to the supplier who sells it. Similarly, a supplier may not have a uniform delivery time; delivery times may vary in relation to the product being delivered. Any attributes that are dependent on the relationship would be associated with the relationship‘s bridge entity.

Recursive Relationships

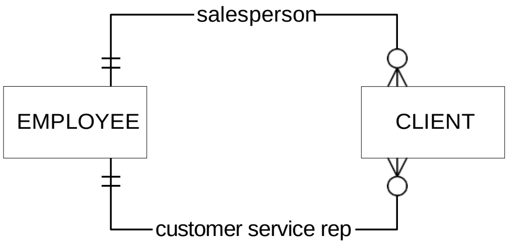
Instances of entities may have relationships with other instances of the same entity. These relationships may be drawn with relationship lines that begin and end connected to the same entity. Common occurrences of these recursive relationships include parent/child relationships:



The diagram above indicates that a person may be the father of zero or many persons, and that a person may have zero or one father. (Not every person's father will be recorded in the system, so the relationship is modeled as optional).

Multiple Relationships Between Entities

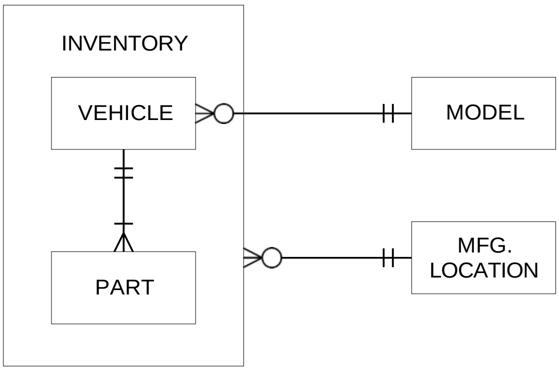
An entity may have multiple relationships with another entity. These are depicted in the ERD with multiple relationship lines connecting the two entities



The diagram above indicates that an employee may be the salesperson assigned to zero or many clients, and an employee may be the customer service representative for zero or many clients. Each client has exactly one salesperson and exactly one customer service representative. Each client's salesperson may or may not be the same employee as the client's customer service representative; each relationship is treated independently.

Entity Subtypes

There are times when it is convenient to depict relationships that apply to an entire class of things, as well as depict relationships that apply only to certain types of the larger class. Entity subtypes accommodate these relationship depictions. In the ERD, entity subtypes may be depicted by entity boxes shown within larger entity boxes. The large entity represents the class, and the smaller boxes depict the subtypes



The example above depicts an ―inventory‖ entity, with the subtypes of ―vehicle‖ and ―part.‖ A vehicle has one or many parts, and every part is associated with one and only one kind of vehicle (according to this diagram, there are no interchangeable components). All items in inventory, whether they are vehicles or parts, have a manufacturing location, but only vehicles are of a particular model.

Structured charts

Using Structure Charts to Design Modular Systems

Once the top-down design approach is taken, the modular approach is useful in programming. This approach involves breaking the programming into logical, manageable portions, or modules. This kind of programming works well with top-down design because it emphasizes the interfaces between modules and does not neglect them until later in systems development. Ideally, each individual module should be functionally cohesive so that it is charged with accomplishing only one function.

Modular program design has three main advantages.

1. Modules are easier to write and debug because they are virtually self-contained. Tracing an error in a module is less complicated, because a problem in one module should not cause problems in others.

2. Modules are easier to maintain. Modifications usually will be limited to a few modules and will not be spread over an entire program. A third advantage of modular design is that modules are easier to grasp, because they are self contained subsystems. Hence, a reader can pick up a code listing of a module and understand its function.

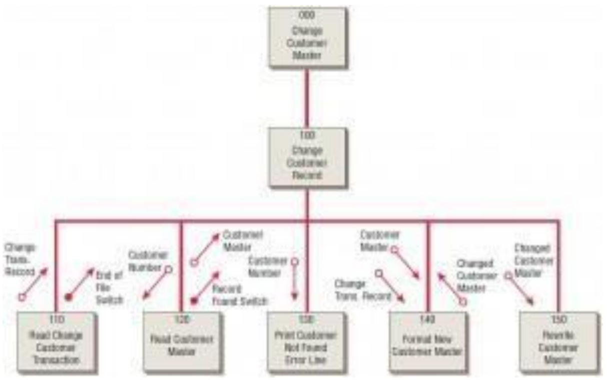
Some guidelines for modular programming include the following:

1. Keep each module to a manageable size (ideally including only one function).
2. Pay particular attention to the critical interfaces (the data and control variables that are passed to other modules).

3. Minimize the number of modules the user must modify when making changes.
4. Maintain the hierarchical relationships set up in the top-down phases.

The recommended tool for designing a modular, top-down system is called a structure chart. A structure chart is simply a diagram consisting of rectangular boxes, which represent the modules, and connecting arrows.

Figure illustrated below is a set of modules used to change a customer record, shows seven modules that are labeled 000, 100, 110, 120, and so on. Higher-level modules are numbered by 100s, and lower-level modules are numbered by 10s. This numbering allows programmers to insert modules using a number between the adjacent module numbers. For example, a module inserted between modules 110 and 120 would receive number 115.



A structure chart encourages top-down design using modules.

Off to the sides of the connecting lines, two types of arrows are drawn. The arrows with the empty circles are called data couples, and the arrows with the filled-in circles are called control flags or switches. A switch is the same as a control flag except that it is limited to two values: either yes or no. These arrows indicate that something is passed either down to the lower module or up to the upper one.

Ideally, the analyst should keep this coupling to a minimum. The fewer data couples and control flags one has in the system, the easier it is to change the system. When these modules are actually programmed, it is important to pass the least number of data couples between modules.

Even more important is that numerous control flags should be avoided. Control is designed to be passed from lower-level modules to those higher in the structure. On rare occasions, however, it will be necessary to pass control downward in the structure. When control is passed downward, a low-level module is allowed to make a decision, and the result is a module that performs two different tasks. This result violates the ideal of a functional module: It should perform only one task.

Even when a structure chart accomplishes all the purposes for which it was drawn, the structure chart cannot stand alone as the sole design and documentation technique. First, it doesn‘t show the order in which the modules should be executed (a data flow diagram will accomplish that). Second, it doesn‘t show enough detail (Structured English will accomplish that)

- **Other**

System development methodologies

There are several approaches or methodologies that can be used. Among them include:

- **structured**

The aim of the structured approach is to build a new and better system and not just improve the old system. It is essentially a team approach and therefore applicable to large project. The approach make use of data flow diagrams and structure charts with accompanying documentation to define a logical method of what is required, leaving the how implementation until later.

The basic approach is:

a) investigate the current system and see what it can achieve,

b) together with any new requirements, produce a logical model of the full specification. This model specifies what is required.

NOTE:

The specifications done in a structured approach needs to be concise, easy to amend and capable of top-down development. That is, we start with a whole problem and identify within it, component parts and then parts of the parts etc. This builds a hierarchy of details down to the most basic level. In order to identify the parts, we use a data flow diagrams and from this we derive structure charts, then individual modules or component of the structure charts can be written up in design language called PSEUDO codes or Structured English.

- **Traditional method/approach**

The traditional approach involves computerizing an improvement on the existing system. Many of the defects are carried over to the new system and there is no allowance for user involvement

- **object oriented design**

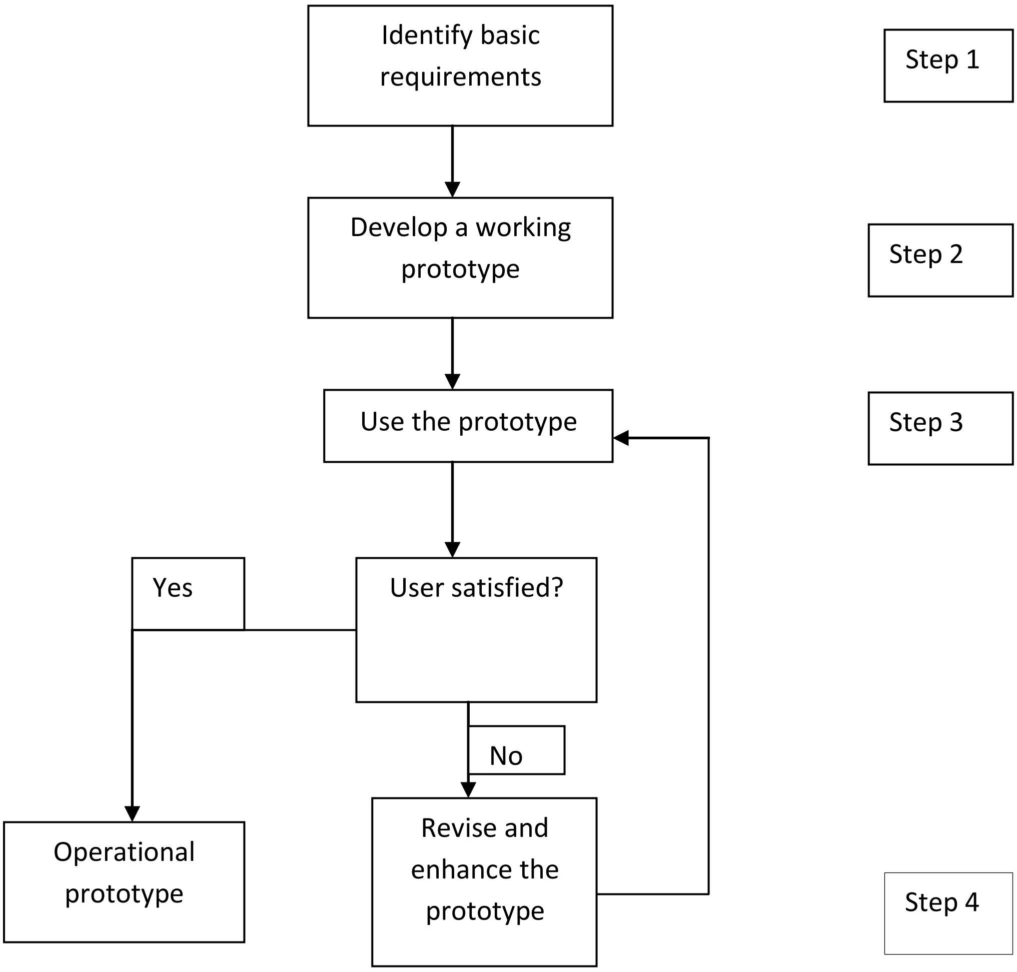
System design methods

Prototyping

Prototyping consists of building an experimental system rapidly and inexpensively for end users to evaluate. By interacting with the prototype users, you can get a better idea of their information requirements. The prototype endorsed by the users can be used as a template to create the final system.

The prototype is a working version of an information system or part of the system but it is meant to be only a preliminary model. Once operational the prototype will be further refined until it conforms precisely to users requirements. Once the design has been finalized the prototype can be converted to a polished production system.

The process of building a preliminary design, trying it out, refining it and trying it again has been called an iterative process of systems development because the steps required to build the system can be repeated over and over again and it actively promotes system design changes. It has been said that prototyping replaces unplanned rework with planned iteration with each version more accurately reflecting user's requirements.


Identify basic requirements
Step 1
Develop a working prototype
Step 2
Use the prototype
Step 3
Yes
User satisfied?
No
Operational prototype
Revise and enhance the prototype
Step 4

The four-step model of the prototyping process,

Step 1 Identify the users basic requirements. The system designer (usually an information systems specialist) works with the user only long enough to capture their basic needs.

Step 2 Develop an initial prototype. The system designer creates a working prototype quickly using 4th generation software, interactive multimedia or computer aided software engineering (CASE) tools.

Step 3 Use the prototype. The user is encouraged to work with the system in order to determine how well the prototype meets their needs and to make suggestions for improving the prototype.

Step 4 Revise and enhance the prototype. The system builder notes all the changes the user requests and refines the prototype accordingly. After the prototype has been revised the cycle returns to step 3 and steps 3 and 4 are repeated until the user is satisfied.

Advantages and Disadvantages

Prototyping is most useful when there is some uncertainty about requirements or design solutions. Prototyping is especially useful in designing information systems end user interface – the part of the system that the users interact with. Because prototyping encourages intense end-user involvement throughout the systems development process it is more likely to produce systems that fulfill user requirements.

However rapid prototyping can gloss over essential steps in systems development. If the completed prototype works reasonably well management may not see the need for reprogramming, redesign or full documentation and testing to build a polished production system. Some of these hastily constructed systems may not easily accommodate large quantities of data or a large number of users in a production environment.

- **Jackson System Development (JSD)**

Jackson System Development (JSD) is a method of system development that covers the software life cycle either directly or, by providing a framework into which more specialized techniques can fit. Jackson System Development can start from the stage in a project when there is only a general statement of requirements. However, many projects that have used Jackson System Development actually started slightly later in the life cycle, doing the first steps largely from existing documents rather than directly with the users.

From the technical point of view there are three major stages in Jackson System Development, each divided into steps and sub-steps:

JSD: The Modeling Stage

In the modeling stage the developers make a description of the aspects of the business or organization that the system will be concerned with. To make this a description they must analyze their business, choosing what is relevant and ignoring what is not. They have to consider the organization as it will be, not as it is now.

The model description is written very precisely. This precision forces the developer to ask detailed questions. It encourages good communication and understanding between developers, users, and everyone else involved with the new system.

The model description consists of actions, entities and related information. An action is an event, usually in the external reality, that is relevant to the system and whose occurrence the system must record. In implementation terms, actions might cause database updates. We start Jackson System Development by making a list of actions with definitions and associated attributes. Diagrams describe ordering relationships between actions. The diagrams describe the entities, people or, things that the system is concerned with.

The data that is to be stored for each entity is then defined. In effect we are choosing what is to be remembered by each entity about the actions that affect it. The full definition of this data includes an elaboration of the entity diagram to show in detail the update rules.

The result of the modeling stage is a set of tables, definitions and diagrams that describe:

- in user terms exactly what happens in the organization and what has to be recorded about what happens, and
- in implementation terms, the contents of the database, the integrity constraints and the update rules.

JSD: The Network Stage

In the network stage we build up a precise description of what the system has to do, including the outputs that are to be produced and the way the system is to appear to the user. This description is in terms of a network of programs. We start this network by making one program for each of the entities that was defined during the modeling stage. The network is then built up incrementally by adding new programs and connecting them up to the existing network. New programs are added for the following reasons:

- To collect inputs for actions, check them for errors, and pass them to the entity programs. In this way entity programs are kept up-to-date with what's happening outside;
- To generate inputs for actions that do not correspond to external events. Such actions are substitutes for real world events, perhaps because those events cannot be detected;
- To calculate and produce outputs.

There are two means of connecting programs in the network. These are by data streams (represented on our network diagram of circles) and by state vector inspection (represented on our network diagrams by diamonds). Whatever kind of connection is appropriate, the entity programs play a pivotal role in the construction of the network. Most of the new programs can be connected directly to the entity programs.

We draw a whole set of network diagrams to describe the system. Different networks usually only have entity programs in common. The complete system is represented by the overlay of all the diagrams.

The diagrams are supported by textual information describing the contents of the data streams and state vector connections. The new programs that are added to the network are defined using the same diagrammatic notation used to describe the ordering of actions. These new programs are designed using the JSP (Jackson Structured Programming) method, which is now a subset of JSD.

JSD: The Implementation Stage

The result of the implementation stage is the final system. This stage is the only one directly concerned with the machine and the associated software on which the system is to run. Therefore, as well as producing and testing code, the implementation stage covers physical design issues. In particular it covers:

- physical data design, and
- reconfiguring the network by combining programs.

Physical data design is about the design of files or databases. The details of database design depend on the particular DBMS being used. However, the necessary information about the application is all available from the network stage. The most important is the data defined for each entity and the high volume accessing of that data as defined by the frequently used state vector connections.

The result of the network stage is a highly distributed network of programs. Often, for convenience or efficiency, we convert programs into subroutines, in effect combining several programs into one, so that a fragment of the network is implemented as a single program. The network is reconfigured from a form appropriate for specification into a form appropriate for implementation.

SSADM – Structured Systems Analysis and Design Method

SSADM is one of the most mature and widely used methods. However it requires a significant investment in training and learning.
This rather simplistic example illustrates the need to specify requirements before the construction of a house (or system) is started. Although it may seem that the requirements are very straight forward constraints may be missed or interdependencies overlooked during development. A problem is far cheaper to put right early in the process that leaving it until the final day before implementation. The systems analyst takes a similar role to that of the architect – a communicator between the client and the builder. Some of the underlying principles of systems analysis, which are also principles of SSADM, help make sure that the user requirements are fully specified.

User involvement –It is a basic principle of SSADM that the users have involvement in and commitment to the development of their system from a very early stage. By ensuring the specification and design match the users requirements at each stage of the analysis and design the risks of producing the –wrong‖ system are very much reduced and possible problems can be sorted out before they become unmanageable.

Quality assurance –In SSADM formal quality assurance reviews are held at the end of each stage where the user is asked to –sign off‖ the design so far. The end products for the stage are scrutinized for quality, completeness, consistency and applicability by the users, developers and by experienced systems staff external to the project.

Separation between logical and physical specifications –SSADM separates logical design from physical design. A hardware/software independent logical design is produced which can easily be translated into an initial physical design. This helps the developers to address one problem at a time and prevents needless constraints at too early a stage in the development. This also helps communication with the users who may not be computer literate but are perfectly able to validate a logical specification or design of their systems.

Why use a structured method?

Structured method share the following characteristics

- They structure a project into small well-defined activities and specify the sequence and interaction of these activities.
- They use diagrammatic and other modeling techniques to give a more precise (structured) definition that is understandable by both users and developers.

Advantages of structured methods – as opposed to a systems analyst using their experience to just ask all the right questions!

- **Structured analysis provides a clear requirements statement** that everyone can understand and is a firm foundation for subsequent design and implementation. Part of the problem with a systems analyst –just asking the right questions‖ is that it is often difficult for a technical person to describe the system concepts back to the user in terms that the user can understand. Structured methods generally include the use of easily understood non-technical diagrammatic techniques. It is important that these diagrams do not contain computer jargon and technical detail that the user won‘t understand – and does not need to understand.
- **More effective use of experienced and inexperienced staff.** A structured method does not remove the need for experienced staff but it does provide the option of spreading the experience more thinly. The use of structured techniques means that certain tasks can be delegated to inexperienced staff who can then be guided by the more experienced.
- **Improved project planning and control.** The use of a structured approach allows the more effective management of projects. Splitting a project down into stages and steps allows better estimation of the time taken to complete a project. Also by following a detailed plan it will be possible to detect slippage as it occurs and not just before the system is due to be implemented.
- **Better quality systems**. By making the specifications very comprehensive it is possible to ensure that the system built will be of a high quality. The use of structures techniques has been found to lead to a system that is very flexible and amenable to change. Within SSADM user participate in formal quality assurance reviews and informal walkthroughs and –sign off‖ each stage before the

developers progress to the next. This means that the analysts can be confident that the new system will meet the users requirements before it is built.

Why choose SSADM?

One of the main advantages is that SSADM builds up several different views of the system that are used to cross check one another. In the building example earlier to help the customer visualize the final building the architect drew several different representations – a cross sectional view, artists impression etc. This probably helped the architect to validate the plans as he made sure that each view was consistent with the others. In SSADM three different views of the system are developed in analysis. These views are closely related to each other and are crosschecked extensively for consistency and completeness. The equal weight given to these three techniques and the prescriptive procedures for checking them against one another is a great strength of the SSADM approach. The three views are,

- Underlying structure of the systems data (Logical data structure)
- How data flows in and out of the system and is transformed within the system (data flow diagrams)
- How the system data are changed by events over time (entity life histories).

Another advantage of SSADM over a number of methods is that it combines techniques into a well-established framework and so as well as providing the techniques for the analyst it gives guidance on how and when to use them. Even though SSADM adopts this rather prescriptive approach there is still a large amount of flexibility within the method and it should be tailored to specific project circumstances.

Basic principles of SSADM

SSADM is a data driven model – this means that there is a basic assumption that the systems have an underlying, generic data structure which changes very little over time, although processing requirements may change. Within SSADM this underlying data structure is modelled from an early stage. The representation of this data structure is checked against the processing and reporting requirements and finally built into the systems architecture.

The structured techniques of SSADM fit into a framework of steps and stages each with defined inputs and outputs. Also there are a number of forms and documents that are specified which add information to that held within the diagrams. Thus SSADM consists of three important features

- Structures - define frameworks of steps and stages and their inputs and outputs.
- Techniques – define how the steps and tasks are performed.
- Documentation – defines how the products of the steps are presented.

Each module is designed to be self contained withthe idea that projects might choose to use SSADM for one module and not for others. SSADM is divided into 5 modules and each of these are divided into stages. Each stage is divided into steps.

Requirements Analysis

Stage 1 – Investigation of current environment.

The current system is investigated for several reasons

- The analysts learn the terminology and function of the users environment
- The old system may form the basis of the new system
- The data required by the system can be investigated
- It provides the users with a good introduction to the techniques
- The boundaries of the investigation can be clearly set.

The current system view built in stage 1 is redrawn to extract what the system does without any indication of how it is achieved. The resulting picture is the logical view of the current system. This allows the analyst to concentrate on what functions are performed in the current system and to make decisions about what must be included in the new system.

Stage 2 – Business system options

They reflect different ways (options) in which the system might be organised to meet the requirements. A decision is made by the users as to which option or combination of options best meets their needs.

Requirements Specification

Stage 3 – Definition of requirements

Based upon the selected Business Systems option, a detailed specification of the required system is built up and checked extensively. In order that the new system will not be constrained by the current implementation there are a number of steps within this stage to lead the analysts gradually away from the current system towards a fresh view of the requirements.

This stage builds up the data design so that all the required data will be included. Is applies a relational analysis technique to groups of data items in the system to act as a cross check on the data definitions.

Logical Systems Specifications

The two stages in this module are often performed

simultaneouslyStage 4 – Selection of technical options

At this stage the development team have enough information to compile the different implementation options for the system. Each option is costed out and the benefits weighed out against the costs to give the user some help in choosing the final solution. This might form the basis for selecting the final system hardware.

Stage 5 – Logical design

The specification developed in stage 3 is expanded to a very high level of detail so that the constructor can be given all the detail necessary to build the system

Physical design

Stage 6 – Physical design

The complete logical design – both data and processing – is converted into a design that will run on the target environment. The initial physical design is tuned before implementation so that it will meet the requirements of the system.

- **Functional decomposition**

System Development Methodologies.

A system development methodology refers to the framework that is used to structure, plan, and control the process of developing an information system. A wide variety of such frameworks have evolved over the years, each with its own recognized strengths and weaknesses. One system development methodology is not necessarily suitable for use by all projects. Each of the available methodologies is best suited to specific kinds of projects, based on various technical, organizational, project and team considerations.

Criteria for choosing system development methodologies

TOPIC 8: SYSTEM IMPLEMENTATION

THEORY

16.2.2.8. T0 Specific Objectives

By the end of this topic, the trainee should be able to:

a) explain the meaning and importance of system implementation

b) explain procedures of system implementation

c) describe system implementation techniques

d) describe system testing techniques

e) explain the levels of acceptance testing

f) explain the need for user training

g) explain the methods of user training

h) describe types of users to be trained

CONTENT

Meaning and importance of system implementation

Systems implementation is the construction of the new system and the delivery of that system into production (that is, the day-to-day business or organization operation).

The Construction Phase of Systems Implementation

The construction phase does two things:

- builds and tests a functional system that fulfils business or organizational design requirements, and
- implements the interface between the new system and the existing production system.

The project team must construct the database, application programs, user and system interfaces, and networks. Some of these elements may already exist in your project or be subject to enhancement.

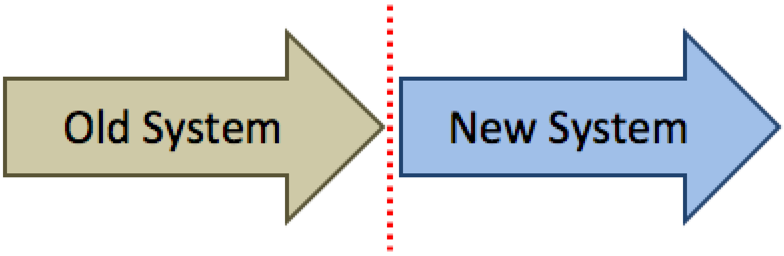
Procedure of system implementation

System implementation techniques

The implementation of the new system occurs when the old system is replaced by the new one. There are mainly four types of implementation techniques applied in systems

1 Direct Changeover

The **old** system is **stopped completely**, and the **new** system is **started**. All of the data that used to be input into the old system, now goes into the new one.



Advantages...

- Takes the **minimal time and effort**
- The **new** system is up and running **immediately**

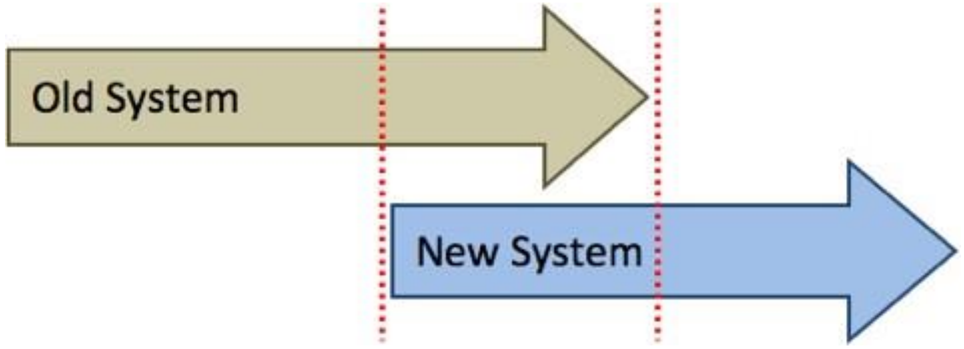
Disadvantages...

- If the new system fails, there is **no back-up system**, so data can be lost

Sometimes a direct changeover is the only way to implement a new system. E.g. an aircraft's auto-pilot system can't have a new and old version running side-by-side, arguing about what to do!

2 Parallel Running

The **new** system is **started**, but the **old** system is **kept running in parallel** (side-by-side) for a while. All of the data that is input into the old system, is **also** input into the new one. Eventually, the old system will be stopped, but only when the new system has been proven to work.



Advantages...

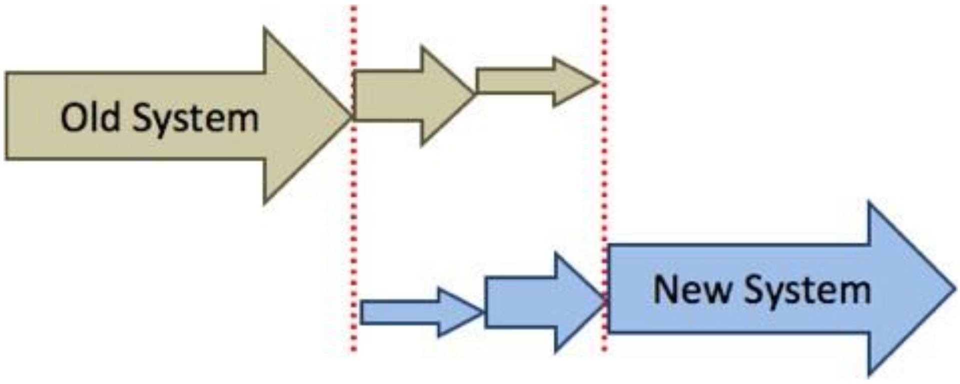
- If anything goes wrong with the new system, the **old system will act as a back-up**.
- The **outputs** from the old and new systems can be **compared** to check that the new system is running correctly

Disadvantages...

- Entering data into two systems, and running two systems together, takes a **lot of extra time and effort**

3 Phased Implementation

The new system is introduced in **phases** (stages, or steps), gradually replacing **parts** of the old system until eventually, the new system has taken over.



Advantages...

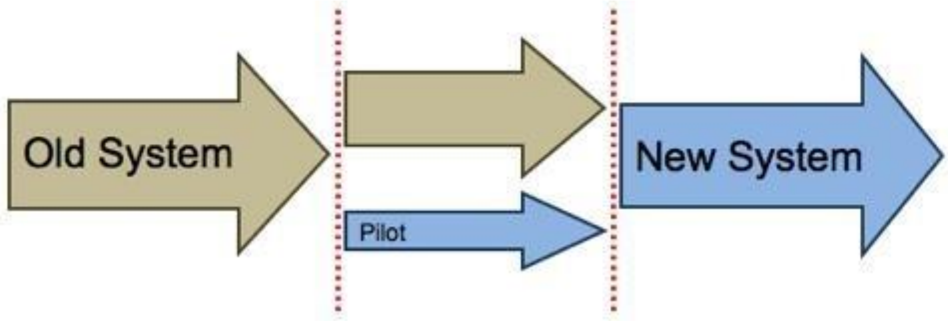
- Allows users to **gradually get used to** the new system
- Staff training can be done in **stages**

Disadvantages...

- If a part of the new system fails, there is **no back-up system**, so data can be lost

4 Pilot Running

The new system is first of all **piloted** (trialled) in **one part** of the business / organisation (e.g. in just one office, or in just one department). Once the **pilotsystem** is running **successfully**, the new system is introduced to **all** of the business / organisation.



Advantages...

- **All features** of the new system can be fully trialled
- If something goes **wrong** with the new system, only a **small part** of the organisation is affected
- The staff who were part of the pilot scheme can help **train** other staff.

Disadvantages...
For the office / department doing the pilot, there is **no back-up** system if things go wrong

System testing and techniques

System testing is the type of testing to check the behaviour of a complete and fully integrated software product based on the software requirements specification (SRS) document. The main focus of this testing is to evaluate Business / Functional / End-user requirements.

This is black box type of testing where external working of the software is evaluated with the help of requirement documents & it is totally based on Users point of view. For this type of testing do not required knowledge of internal design or structure or code.

In the integration testing, testers are concentrated on finding bugs/defects on integrated modules. But in the ***Software System Testing*** testers are concentrated on finding bugs/defects based on software application behavior, software design and expectation of end user.

Why system testing is important:

a) In Software Development Life Cycle the System Testing is performed as the first level of testing where the System is tested as a whole.

b) In this step of testing check if system meets functional requirement or not.

c) *System Testing* enables you to test, validate and verify both the Application Architecture and Business requirements.

d) The application/System is tested in an environment that particularly resembles the effective production environment where the application/software will be lastly deployed.

Generally, a separate and dedicated team is responsible for **system testing**. And, System Testing is performed on staging server which is similar to production server. So this means you are testing software application as good as production environment.

Different Hierarchical levels of testing:

As with almost any technical process, software testing has a prescribed order in which things should be done. Different levels of testing are used in the testing process; each level of testing aims to test different aspects of the system. The following is lists of software testing categories arranged in sequentially organize.

Hierarchical levels of testing

Unit testing – Testing is done in the development process while developer completes the unit development. The objective of this testing is to verify correctness of the module. The purpose of unit testing is to check that as individual parts are functioning as expected. Basically Unit testing is typically carried out by the developer.

Integration testing – *System Integration Testing* is started after the individual software modules are integrated as a group. A typical software project consists of multiple modules & these are developed by different developers. So in integration testing is focuses to check that after integrating modules Is two modules are communicating with each other or not. It is critical to test every module‗s effect on the entire program model. Most of the issues are observed in this type of testing.

System testing – This is the first time end to end testing of application on the complete and fully integrated software product before it is launch to the market.

Acceptance testing – User acceptance is a type of testing performed by the Client to certify the system with respect to the requirements that was agreed upon. This is beta testing of the product & evaluated by the actual end users. The main purpose of this testing is to validate the end to end business flow

Acceptance testing

What is User Acceptance Testing?

User Acceptance testing is the software testing process where the system is tested for acceptability. Such type of testing executed by client in separate environment (similar to production environment) & confirm whether the system meets the requirements as per requirement specification or not.

UAT is performed after System Testing is done and all or most of the major defects have been fixed. This testing is to be conducted in the final stage of Software Development Life Cycle (SDLC) prior to system being delivered to a live environment

The **Acceptance testing** is **–black box‖** tests, means UAT users are not aware of internal structure of the code, they just specify the input to the system & check whether systems respond with correct result.

Levels of testing – User Acceptance Testing

User Acceptance testing also known as Customer Acceptance testing (CAT)

The CAT or UAT are the final confirmation from the client before the system is ready for production. The business customers are the primary owners of these UAT tests. It is a collaboration between business customers, business analysts, testers and developers. It consists of test suites which involve multiple test cases & each test case contains input data (if required) as well as the expected output. The result of test case is either a pass or fail.

Prerequisites of User Acceptance Testing:

Prior to start the UAT following checkpoints to be considered:

- The Business Requirements should be available.
- The development of software application should be completed & different levels of testing like Unit Testing, Integration Testing & System Testing is completed.
- All High Severity, High Priority defects should be verified. No any Showstoppers defects in the system.
- Check if all reported defects should be verified prior to UAT starts.
- Check if Traceability matrix for all testing should be completed.
- Before UAT starts error like cosmetic error are acceptable but should be reported.
- After fixing all the defects regression Testing should be carried out to check fixing of defect not breaking the other working area.
- The separate UAT environment similar to production should be ready to start UAT.
- The Sign off should be given by System testing team which says that Software application ready for UAT execution.

TYPES OF ACCEPTANCE TEST

A] Alpha Testing

Alpha testing is conducted by Customer at the developer‗s site, it is performed by potential users like developer, end users or organization users before it is released to external customers & report the defects found while Alpha testing.
This software product testing is not final version of software application, after fixing all reported bug (after bug triage) the new version of software application will release. Sometimes the Alpha Testing is carried out by client or an outsider with the attendance of developer and tester. The version of the release on which Alpha testing is perform is called –**Alpha Release**‖.

B] Beta Testing

Most if times we have the sense of hearing term –Beta release/version‖, so it is linked to Beta Testing.

Basically the beta testing is to be carried out without any help of developers at the end user‗s site by the end users &, so it is performed under uncontrolled environment. Beta testing is also known as Field testing. This is used to get feedback from the market.

This testing is conducted by limited users & all issues found during this testing are reported on continuous basis which helps to improve the system. Developers are taking actions on all issues reported in beta testing after bug triage & then the software application is ready for the final release. The version release after beta testing is called –**Beta Release–**.
This is last stage of the testing where product is sent outside the company or for trial offer to download.

What to Test in User Acceptance Testing?

- Based on the Requirements definition stage use cases the Test cases are created.

- Also the Test cases are created considering the real world scenarios for the application.
- The actual testing is to be carried out in environments that copy of the production environment. So in the type of testing is concentrating on the exact real world use of application.
- Test cases are designed such that all area of application is covered during testing to ensure that an effective User Acceptance Testing.

What are the key deliverable of User Acceptance Testing?

The completion of User Acceptance Testing is the significant milestone for traditional testing method. The following key deliverable of User Acceptance Testing phase:

- **Test Plan**: This outlines the Testing Strategy
- **UAT Test cases**: The Test cases help the team to effectively test the application in UAT environment.
- **Test Results and Error Reports**: This is a log of all the test cases executed and the actual results.
- **User Acceptance Sign-off**: This is the system, documentation, and training materials have passed all tests within acceptable margins.
- **Installation Instructions**: This is document which helps to install the system in production environment.
- **Documentation Materials**: Tested and updated user documentation and training materials are finalized during user acceptance testing

Need for user training

So you‘ve made the decision, selected your software, implementation is underway and you are shortly due to go live. So much to organise and yet often the crucial process of incorporating an end-user strategy is forgotten – arguably the most important. Unless your staff are trained on using your new software, you cannot expect to reap the full benefits of the investment.

In addition to this there is often the challenge of staff being resistant to change. By explaining the reasons behind a new system to create an understanding, outlining the benefits and providing training, you will be on the road to staff embracing these changes.

The key to planning the end-user strategy is reviewing the practicalities – How many people need to use and understand the new system? How quickly do you want the rollout to all staff?

Once you have an idea of the above, you can decide on the most appropriate delivery of training. You may select hands-on computer training with an instructor, seminar style with a presenter, manual based training or maybe even virtual learning. you may also need to consider to conduct the training individually or in group focused sessions.

Selecting the best delivery method will depend on several factors:

- The number of staff being trained and the timescale of delivery – this will determine whether you train as a group or individually
- How your staff are geographically spread – finding the best location to conduct the training and consider virtual learning
- The processes involved in the new software that staff need to grasp, for example maybe practical training through the activity will speed up the learning process as opposed to a presentation on how to use.
- What resources are required for the training – Uploading software or using the internet, manual production?
- The level of Interactivity required eg. – will role play be necessary, would group activity increase understanding and raise motivation levels?
- Who is best to facilitate and deliver training – do the software company recommend or provide training for staff or will you require an external/internal facilitator?

Creating the correct training environment is another consideration. One aspect of this is bridging the gap between staff skill sets, not all staff might need the same level of training, it could be more effective learning and time-wise to deliver tiered-skill level training to separate groups.

Following this, the physical learning environment chosen will impact learning. Staff must be comfortable, relaxed, have the necessary tools and no distractions in order to listen, interpret and actively engage in the information being passed on to them. So often, IT and technical issues can get in the way of learning; the smooth delivery of training is key to those receiving and eliminates frustration of those delivering. By getting this right, you will save time and money in unnecessary errors being made. To maintain the benefits of your new system, it is equally important to educate new starters, possibly consider incorporating it into Induction plans and for those fully trained, updates and refresher training should be scheduled.

Methods of user training

Types of users to be trained

TOPIC 9: SYSTEM MAINTENANCE AND REVIEW

THEORY

16.2.2.9. T0 Specific Objectives

By the end of this topic, the trainee should be able to:

a) explain the meaning of system maintenance and review

b) explain the importance of system maintenance

c) describe the types of system maintenance

CONTENT

Meaning and importance of system maintenance and review

Importance of system maintenance

Types of system maintenance

TOPIC 10: SYSTEM DOCUMENTATION

THEORY

16.2.2.10. T0 Specific Objectives

112 By the end of this topic, the trainee should be able to:

a) explain the meaning of system documentation

b) explain the need for system documentation

c) describe the types of system documentation

CONTENT

Meaning of system documentation

In computer hardware and software product development, documentation refers to: **The information that describes the product to its users.** It consists of the product technical manuals. The term is also sometimes used to mean the *source* information about the product contained in design documents, detailed code comments, etc

The term is derived from the idea that engineers and programmers "document" their products in formal writing. The earliest computer users were sometimes simply handed the engineers' or programmers' "documentation." As the product audience grew, it became necessary to add professional technical writers and editors to the process. In this task-oriented view, product information can be divided into and sometimes physically organized into these task categories: evaluating, planning for, setting up or installing, customizing, administering, using, and maintaining the product. Documentation is now often built directly into the product as part of the user interface and in help pages.

Functional specification

Definition

A functional specification is a formal document used to describe in detail for software developers a product's intended capabilities, appearance, and interactions with users.

The functional specification is a kind of guideline and continuing reference point as the developers write the programming code. Typically, the functional specification for an application program with a series of interactive windows and dialogs with a user would show the visual appearance of the user interface and describe each of the possible user input actions and the program response actions. A functional specification may also contain formal descriptions of user tasks, dependencies on other products, and usability criteria.

For a sense of where the functional specification fits into the development process, here are a typical series of steps in developing a software product:

- **Requirements**. This is a formal statement of what the product planners informed by their knowledge of the marketplace and specific input from existing or potential customers believe is needed for a new product or a new version of an existing product. Requirements are usually expressed in terms of narrative statements and in a relatively general way.
- **Objectives**. Objectives are written by product designers in response to the Requirements. They describe in a more specific way what the product will look like. Objectives may describe architectures, protocols, and standards to which the product will conform. *Measurable objectives* are those that set some criteria by which the end product can be judged. Objectives must recognize time and resource constraints.
- **Functional specification**. The functional specification (usually *functional spec* or just *spec* for short) is the formal response to the objectives. It describes all *external* user and programming interfaces that the product must support.
- **Design change requests**. Throughout the development process, as the need for change to the functional specification is recognized, a formal change is described in a design change request.
- **Logic specification**. The logic specification describes *internal interfaces* and is for use only by the developers, testers, and, later, to some extent, the programmers that service the product and provide code fixes to the field.
- **User documentation**. In general, all of the preceding documents (except the logic specification) are used as source material for the technical manuals and online information (such as help pages) that are prepared for the product's users.

- **Test plan**. Most development groups have a formal test plan that describes test cases that will exercise the programming that is written. Testing is done at the module (or unit) level, at the component level, and at the system level in context with other products. This can be thought of as *alpha testing*. The plan may also allow for beta test. Some companies provide an early version of the product to a selected group of customers for testing in a "real world" situation.
- **The final product**. Ideally, the final product is a complete implementation of the functional specification and design change requests, some of which may result from formal testing and beta testing.

The cycle is then repeated for the next version of the product, beginning with a new Requirements statement, which ideally uses feedback from customers about the current product to determine what customers need or want next.

Most software makers adhere to a formal development process similar to the one described above.

Need for system documentation

It helps to provide the ***clear description*** of the work done so far. It is essential that the documents prepared must be updated on regular basis this will help to trace the progress of work easily. With appropriate and good documentation it is very easy to understand the how aspects of the system will work for the company where the system is to installed. It also help to understand the type of data which will be inputted in the system and how the output can be produced.

After the system is installed, and if in case the system is not working properly it will be very easy for the administrator to understand the flow of data in the system with documentation which will help him/ her to correct the flaws and get the system working in no time.

Uses of Documentation

- It facilitates effective communication regarding the system between the technical and the non technical users.
- It is very useful in training new users. With a Good documentation new users can easily get acquainted with the flow of the systems.
- Documentation also helps the users to solve problems like trouble shooting even a non technical user can fix the problems.
- It plays a significant role in evaluation process.
- It not only helps to exercise a better control over the internal working of the firm, but it also external as well especially during audit.
- Documentations can help the manager to take better financial decisions of the organization.

The Value of Documentation

Documentation is valuable as reference and standardization material **and** because it creates value for organizations and products:

- Good documentation tends to reduce the volume of support requests and erroneous bug/issue reporting. Additionally, good comprehensive documentation makes responding to the remaining support requests easier.
- Good documentation gets knowledge out of people‘s heads and into a shared format. This increases reliability, because it reduces a dependence on people‘s memories and notes, which may not always be accessible, or may be inconsistent.
- In absence of a specification, documentation can help define and regulate the user experience as development proceeds.
- Without documentation users may be unaware of features and behaviors, which reduces their value. If users never learn about or take advantage of new features and functionality, then development resources are essentially wasted.

This doesn‘t mean you should write documentation in situations where organizing a system or environment is more practical and efficient, or that standardizing a process with a script or program doesn‘t have it‘s place. However, for complex problems where users interact with your interfaces or processes, documentation is often *the answer*. Take pride in documentation, treat it seriously, and the return will be great.

Types of system documentation

Sommerville describes two main categories of software documentations; process and product documents.

Process:
documentations are used to manage the development process for example planning, scheduling and cost tracking, standards among others.

Product documentations:
Describe the main deliverable (software product) and some of the documents in this category form part of deliverables. These include;

- ❖ Requirements Specification,
- ❖ Design documents,
- ❖ Commented Source Code,
- ❖ Test Plans including test cases,
- ❖ Validation and Verification plan and results

Application and benefits of documentation
The role of documentation in a software engineering environment is to communicate information to its audience and instill knowledge of the system it describes.

- System documentation plays another role namely software maintenance, and
- Identified several benefits of documentation. These include:

- ➢ easier reuse of old designs,
- ➢ better communication about requirements,
- ➢ more useful design reviews,

- easier integration of separately written modules,
- more effective code inspection,
- more effective testing, and
- more efficient corrections and improvements.

Researchers and practitioners also have looked at the uses of software documentation and just to compile a few, notes that documentation helps at software development, keeps software-quality at high levels and makes it easy to transfer projects.

Documentation can also be used for:

- learning a software system,
- testing a software system,
- working with a new software system,
- solving problems when other developers are unavailable to answer questions,
- looking for big-picture information about a software system
- maintaining a software system
- answering questions about a system for management or customers,
- looking for in-depth information about a software system, and
- facilitates effective communication regarding the system between the technical and the non technical users,
- training new users,
- solve problems like trouble shooting, evaluation process, and quantify the financial ramifications/footprint of the system.

TOPIC 11: SYSTEM ACQUISITION

THEORY

16.2.2.11. T0 Specific Objectives

By the end of this topic, the trainee should be able to:

a) describe information system acquisition methods

b) explain the criteria for choosing an information system acquisition method

CONTENT

Information system acquisition methods

Criteria For Information System Acquisition

There are several methods that can be applied in acquiring information systems for an organization.

Key Factors in Selecting Available Alternatives

Some main alternatives exist in acquiring IT/IS applications. Some major options are **buy, lease, develop in-house, or outsourcing** a system from any other companies. When does it make more sense to buy the applications? When does it make more sense to lease? Do we have to outsource our applications to other companies? The set of processes for the acquisition decision must be identical for every instance or business opportunity that arises. In the past anything that has been deemed strategic has been built in-house but the trend to outsource and buy more systems has grown. Below are some critical factors that should be evaluated prior to choose preferable IS procurement strategy, whether to buy, lease, build in-house, or outsource IT applications.

1. Buying the Applications (Off-the-Shelf Solution)

Purchasing commercially available solutions requires that the business adapt to the functionality of the system. Buying an existing package can be a cost effective and time saving strategy compared with in-house development. The business adaptation process obliges that theorganization could also customize the software product and subsequently maintains those customizations within the processes that have been modified and changed. Most organizations are rarely fully satisfied by one software package. Therefore, it is sometimes necessary to acquire multiple packages to support even one business process. Note that when selecting a vendor package, organizations should consider the following key factors:

• Vendor stability • System upgrades • Customer support provided by vendor	• Hardware and software requirements • Required customization of the base software

A _buy�Ÿ option should be carefully considered to ensure all the critical features of the current and future needs are included in the package. Buying makes sense if an organization plan to keep something for a long time, but technology typically becomes outdates every two to three years. The reason a small-business customer never buys information-technology equipment is because there is an obsolescence curve. When you know something is going to become obsolete, why does a small-business customer want to be an owner of that equipment, instead of simply a user of that equipment? When the business is all about cutting-edge technology, buying can make good sense. Eventually, buying decision typically means picking up something inexpensive to do the job right now.

The advantages and shortcomings of _buy‘ option are summarized in Table2.

TABLE 2. Advantages and Disadvantages of ‘Buy’ Option

ADVANTAGES	DISADVANTAGES
· Shorter implementation time · Use of proven technology · Availability of outside technical expertise · Easier to define costs · Frequent software updates · The price is usually cheaper · Minimal IT personnel	· Incompatibility with company needs · Incompatibility between different applications · Limitation on the software customization · Have no control over software improvements · Long term reliance on vendor support · Specific hardware or software requirements

2. Leasing the Applications

Lease option can result in substantial cost and time savings compared to buy option or in-house development. Leasing can be a good choice for small-medium enterprise that cannot afford large investment in IT applications. Moreover, many common features that are needed by most organizations are usually incorporated in the leased package even though it may not always exactly include all the required features. Also, regarding a shortage of IT personnel, many companies choose to lease instead of develop software in-house. Leasing can help you decrease the total cost of ownership of technology assets. It allows you to track, standardize and regularly upgrade your practice's technology. Therefore, leasing requires another kind of management skill, too, which is: Lifecycle management. Whereas buying typically means picking up something inexpensive to do the job right now, leasing means that a business is looking at the bigger picture, planning for future upgrades and evolving needs. When controlling cash flow is critical and you don't have time to worry about your equipment, leasing can be a great option. Other vendors concur that built-in protections against obsolescence can encourage leasing.

When you enter into a lease, the ability to progress from one generation of technology to the next, to expand your technology solution, to rid yourself of obsolete equipment is far easier and far smoother, because of the way a lease is structured for small and medium businesses.

The advantages and disadvantages of _lease‘ option are summarized in Table 3.

TABLE 3. Advantages and Disadvantages of 'Lease' Option

ADVANTAGES	**DISADVANTAGES**
· Shorter time implementation · Cost saving (cheaper than buy option) · Ease to maintain cash flow · Required only minimum IT staff · Less risky to anticipate technology updates · Having most of the required features	· May not exactly fit with company needs · Limitation on the software customization · Have no control over software improvements · Specific hardware or software requirements · Include an interest component that a cash purchase would not include

3. Developing the applications in-house

Another strategy of IT acquisition is to build the application in-house. This option works well for the organization that has the resources and time to develop the IT applications by its own.

This approach may be time consuming and somehow costly, but the company may have a system that meet all the organization objective requirements. Its overriding advantage was the freedom to create a system that would closely fit the company‘s business processes. In contrast to other solutions, it would be relatively inexpensive; however, it would take more time (maybe significantly more) to implement. If successful, the outcome could lead to another revenue stream from royalties for software sales.

There are two major ways to develop the system in-house.

First, building the application from the scratch is one of the approaches to match the specific application with the requirements.

Another way of building the in-house application is using the standard components or features that have been included in some commercial packages (i.e. Java, Visual Basic, C++) or using available packaged software that can be customized. However, the second approach offers greater flexibility, cost and time saving rather than building the software from the base.

The advantages and limitations of _in-house development‘ option are summarized in Table 4.

TABLE 4. Advantages and Disadvantages of ‘In-house development’ Option

ADVANTAGES	DISADVANTAGES
· Best fit with the company requirements · Have control over software improvements · Have all of the required features · Main core competencies and maintain level of quality service · Make a distinction with other companies	· Required more IT personnel · High overhead cost · Time consuming · Problem with usability of the system · High switching cost · Difficult to update to newer technology

4. Outsourcing the applications

One of the recent trendsetters in IT/IS acquisition strategy is outsourcing. Outsourcing is a strategic use of outside resources to perform activities traditionally handled by internal staff and resources. Laudon and Laudon define it as the process turning over an organization computer center, telecommunication network or application development to external vendors. However, in general, the reasons boiled down to one factor. *It is less costly for the purchasing company to turn* *outside rather than do the work in house*. This strategy does not have to the expertise and it is less costly to buy the expertise than build it. Perhaps it does not have time of pull off a project. However, it can take advantage of the economy of scale that the provider has and which the purchasing company does not. As a matter of fact, the organization turns to outsourcing to save money. Decisions as what to and whether to outsource should be tied to an identification and understanding of an organization‘s core competencies and its critical success factors. If a task is a both a core competency and a critical success factor, it should not be considered outsourcing. Such projects are at the heart of the company. Success or failure of such functions is directly tied to success or failure of the company as a whole. In general, such functions are critical to an organization‘s day-to-day operations, ability to competitively differentiate itself, ability to deliver value to customers and partners, and ability to innovate·

Some different types of outsourcing relationships are partnership, service provider, and vendor. Strategic partnerships might even establish some form of mutual ownership or revenue sharing. A service provider relationship is established when an ASP operate MIS applications and contracts are flexible to respond to changing needs of the organization. Most of ASPs develop custom software applications for the line of business. Meanwhile, vendor or transaction partnerships are more typical outsourcing arrangements where a company simply contracts with a vendor to provide the service or product. The vendor usually provides hardware, telecommunication, backup, and managed applications for the company. In this term of partnership, contracts usually escalate fees based on levels of usage of line item services.

Outsourcing can be utilized to exploit the lower cost base of external service providers, which allows for reduction in operating costs. Having access to work with IT expertise would be another benefit of outsourcing; thus, it reduces the risk of technology obsolescence and overtaking competitors on the technological front. Furthermore, it allows a company to ***focus on its core business*** and reduce its workload. Some limitations of this strategy include the risk of loosing the organizational core competencies, reduction in the quality of service received by a client, and also some risk of the rise of unexpected expenses.

The advantages and shortcomings of the _outsourcing' option are summarized in Table 5.[23]

TABLE 5. Advantages and Disadvantages of Outsourcing

ADVANTAGES	**DISADVANTAGES**
· Cost Reduction	· Loss of organizational competencies
· Access to word class specialist providers	· Reduction in quality of services
· Improved focus on core business	· Cost escalation from unforeseen expenses

· Subcontracting of workload · Better risk management	

TOPIC 12: ICT PROJECT MANAGEMENT

THEORY

16.2.2.12. T0 Specific Objectives

114 By the end of this topic, the trainee should be able to:

a) explain the meaning and importance of ICT project management

b) describe ICT project management tools

c) explain the criteria for evaluating ICT projects

d) explain the signs of a failing ICT project

e) explain the reasons for ICT project failure

f) explain the strategies for managing a failing ICT project

CONTENT

Meaning and importance of ICT project management

A **project** is a planned series of related activities for achieving a specific business objective. Information systems projects include the:

- development of new information systems,
- Enhancement of existing systems, or upgrade or replacement of the firm‘s information technology (IT) infrastructure.

Project management refers to the application of knowledge, skills, tools, and techniques to achieve specific targets within specified budget and time constraints. Project management activities include:

- planning the work,
- assessing risk,
- estimating resources required to accomplish the work,
- organizing the work,
- acquiring human and material resources
- assigning tasks,
- directing activities,
- controlling project execution,
- reporting progress, and
- analyzing the results.
-

As in other areas of business, project management for information systems must deal with five major variables:

- scope,
- time,
-

cost,

- quality, and
- risk

Scope defines what work is or is not included in a project. For example, the scope of project for a new order processing system might be to include new modules for inputting orders and transmitting them to production and accounting but not any changes to related accounts receivable, manufacturing, distribution, or inventory control systems. Project management defines all the work required to complete a project successfully, and should ensure that the scope of a project does not expand beyond what was originally intended.

Time is the amount of time required to complete the project. Project management typically establishes the amount of time required to complete

major components of a project. Each of these components is further broken down into activities and tasks. Project management tries to determine the time required to complete each task and establish a schedule for completing the work.

Cost is based on the time to complete a project multiplied by the cost of human resources required to complete the project. Information systems project costs also include the cost of hardware, software, and work space. Project management develops a budget for the project and monitors ongoing project expenses.

Quality is an indicator of how well the end result of a project satisfies the objectives specified by management. The quality of information systems projects usually boils down to improved organizational performance and decision making. Quality also considers the accuracy and timeliness of information produced by the new system and ease of use.

ICT project management tools

Project management is a challenging task with many complex responsibilities. Fortunately, there are many tools available to assist with accomplishing the tasks and executing the responsibilities.

Project managers should choose a project management tool that best suits their management style. No one tool addresses all project management needs.

Program Evaluation Review Technique (PERT) and Gantt Charts are two of the most commonly used project management tools. Both of these project management tools can be produced manually or with commercially available project management software.

PERT is a planning and control tool used for defining and controlling the tasks necessary to complete a project. PERT charts and Critical Path Method (CPM) charts are often used interchangeably; the only difference is how task times are computed. Both charts display the total project with all scheduled tasks shown in sequence. The displayed tasks show which ones are in parallel, those tasks that can be performed at the same time. A graphic representation called a "Project Network" or "CPM Diagram" is used to portray graphically the interrelationships of the elements of a project and to show the order in which the activities must be performed.

PERT planning involves the following steps:

1. *Identify the specific activities and milestones.* The activities are the tasks of the project. The milestones are the events that mark the beginning and the end of one or more activities.
2. *Determine the proper sequence of activities.* This step may be combined with #1 above since the activity sequence is evident for some tasks. Other tasks may require some analysis to determine the exact order in which they should be performed.
3. *Construct a network diagram.* Using the activity sequence information, a network diagram can be drawn showing the sequence of the successive and parallel activities. Arrowed lines represent the activities and circles or "bubbles" represent milestones.

4. *Estimate the time required for each activity.* Weeks are a commonly used unit of time for activity completion, but any consistent unit of time can be used. A distinguishing feature of PERT is it's ability to deal with uncertainty in activity completion times. For each activity, the model usually includes three time estimates:
    - **Optimistic time** - the shortest time in which the activity can be completed.
    - **Most likely time** - the completion time having the highest probability.
    - **Pessimistic time** - the longest time that an activity may take.

    From this, the expected time for each activity can be calculated using the following weighted average:

    Expected Time = (Optimistic + 4 x Most Likely + Pessimistic) / 6

5. *Determine the critical path.* The critical path is determined by adding the times for the activities in each sequence and determining the longest path in the project. The critical path determines the total calendar time required for the project. The amount of time that a non-critical path activity can be delayed without delaying the project is referred to as **slack time.**

    If the critical path is not immediately obvious, it may be helpful to determine the following four times for each activity:

    - ES - Earliest Start time
    - EF - Earliest Finish time
    - LS - Latest Start time
    - LF - Latest Finish time

    These times are calculated using the expected time for the relevant activities. The earliest start and finish times of each activity are determined by working forward **(forward pass)** through the network and determining the earliest time at which an activity can start and finish considering its predecessor activities.

The latest start and finish times are the latest times that an activity can start and finish without delaying the project. LS and LF are found by working backward **(backward pass)** through the network. The difference in the latest and earliest finish of each activity is that activity's slack. The critical path then is the path through the network in which none of the activities have slack.

Criteria for evaluation ICT projects

Signs of a failing ICT project

Has your project gone out of control--threatening to be both behind schedule and over budget? The warning signs were probably there from the beginning. Here's how to figure out what went wrong and what you can do to save a failing project.

IS projects that spiral out of control need project managers with skill, experience, and resilience to get them back on track. You may develop a hunch that a project is starting to get out of control when you notice that stakeholders are not attending project meetings, developers are leaving the project, or the financial group is asking just too many questions about daily expenses/resource allocations.

Your challenge as a project or development manager is to proactively identify projects that have gone bad and then decide whether to reengineer (i.e., retrofit) those problems. Bad projects, left untouched, could cause irreparable harm to any organization.

Some typical problems you may encounter on a project, together with possible solutions, are shown in **Figure A**.

Figure A

Symptom	Solution
Poor communications	Develop a communications plan and inform all stakeholders · Establish frequent project meetings · Document roles & responsibilities
Poor estimation & planning	Use Analysts, SMEs and/or Cost Estimators to assist in estimating
Minimal documentation	Identify the minimum documentation needed on the project · Business Case / Project Brief · User Requirement Specification · Technical Specification(s) as needed · Deployment Plan · Project Schedule · Training schedule · Test Plan
Poor project management	Either replace the project manager with a stronger candidate or involve executive direction and leadership
Poor executive buy-in	Obtain immediate executive sponsorship through escalation
Poor user requirements	Document and approve user requirements and establish the necessary change controls to support any future changes

Depending upon the unique aspects of a project, there could be a multitude of reasons for a project going out of control. Risk factors include the following:

- **Sloppy requirements:** Every project depends upon solid user requirements being firmly locked down prior to any work being undertaken. Failure to do so is a leading cause for project failure. Somehow, the trend is that many project teams think they can get started by rushing the requirements-gathering phase. These projects are then eagerly started with incomplete requirements. If you are developing a project using a standard waterfall methodology, any incomplete requirements will have both a negative cost and schedule impact on the project. On iterative development projects, user requirements are still of utmost importance, but can be negotiated ahead of any actual development.

- **Schedule slippage:** Many times, project schedules spiral out of control when dates and deliverables aren‘t aggressively monitored and tracked on a daily basis. All too often, managers leave issues unresolved for days, which then results in schedule overruns. I recommend that that you check project schedules daily.
- **Budget overrun:** Projects that run over budget are sometimes more prone to being cancelled because senior executives are concerned about cash going into and out of company coffers. If a project starts showing gradual cost overruns, the project is often still given a chance, but, as the losses mount and show no sign of recovery, cancelling the project may be necessary. In reality, though, some projects are so critical to business survival that they can‘t be stopped. Therefore, the cost overruns are simply absorbed or skilfully transferred elsewhere. This means that project managers must manage their actual budgets against the planned budget and keep their stakeholders aware of any deviation.
- **Scope creep:** When clients insist on ever-increasing changes to the product being developed, scope creep can jeopardize the project. I don‘t know of too many project managers who can handle too many changes all at once. An even more difficult situation for a project manager surfaces when new changes are introduced after the project has been launched. This usually drives up the cost, resource requirements, deliverables, and completion time. Scope creep needs to be managed and the project manager needs to have a change control process in place to assess the impact and cost of the change and, possibly, negotiate the change for a future release.
- **Poor planning and estimation:** Those projects that are poorly estimated and planned tend to fail both in cost and schedule, which eventually causes the overall project to fail. Managers tend to start projects without relying on proper analysis, and sizing, and fail to consult subject matter experts or cost estimators to validate how much project work packages will cost.
- **Poor documentation:** Maintaining inadequate project documentation is cause for concern and should raise the red flag. Lessons learned from many failed projects reveal that there was too little documentation to adequately describe the project in its broader terms, and serve as a clear communication channel.
- **New technology:** Projects that require integrating new tools and deploying new vendor applications/devices are always far more difficult, because usually, only the vendor engineers clearly understand the limitations and functionality of the products. This results in delays in project schedule and could require weeks or months before the products are stable enough to be deployed. If a project is undertaken using new technology, managers should be aware of the associated risks and plan their schedules accordingly.
- **Poor communications:** One of the biggest reasons why any project goes bad is due to a lack of communication. I have seen many projects fail simply because no one understands what to do and receives no communication as to current progress; this, inevitably, results in project failure.

- **Poor decision-making:** Decision that aren't made at all and decisions that are delayed due to second-guessing are both risk factors. Additionally, some decisions are so turned out-of-context as responsibility for them is passed down the line that they end up being made based on faulty information. This doesn‘t bode too well for any critical project.
- **Poor project management:** The person managing the project may not have the skills or experience to pull it off. Yes, any project can be stuck with a lame duck manager. In such cases, it may be necessary to stop the project, replace the project manager, make the necessary adjustments, and start again. The departing manager should be given the option to provide his/her version of the story, though, before moving on.
- **Poor testing:** A big culprit on any project is having either too little testing or, in many cases—if a test team is involved—testing too late in the process. Both testing and quality assurance need to be built into the project from the day the project is launched.

Reasons for ICT project failure

Strategies for managing a failing ICT project

Control Measures -Steps toward finding the fix

For project managers working against time (i.e., competitive industries), it may be useful to try to get the project back on track by taking the following steps:

- **Read between the lines:**

  Once you start getting wind of problems such as schedule, quality, cost, resource conflicts or late status reports, then I‘d recommend a one-on-one review with the applicable project manager as soon as possible to determine the project‘s true status.

- **Recognize what went wrong:**

  Try and assess what the reasons were for the project failing in the first place. It‘s no use simply blaming people. If the failure was due to bad estimation or planning, then efforts need to be put into place to correct any future projects following the same path.

- **Determine what to do:**

  Should you cancel a bad project or try and bring it back on track? Organizations faced with bad projects should first try and salvage their poor performers rather than cancelling them.

- **Educate your project staff:**

  Training and education really go a long way to correcting bad project practices.

TOPIC: EMERGING TRENDS IN SAD

THEORY

16.2.2.13. T0 Specific Objectives

By the end of this topic, the trainee should be able to:

a) identify emerging trends in SAD

b) explain the challenges of emerging trends in SAD

c) cope with the challenges of emerging trends

CONTENT

Emerging trends in SAD

Challenges of emerging trends in SAD

Cope with the challenges of emerging trends in SAD

ASSIGNMENT

Both the questions each carry 15marks.

Question 1.

a) Describe the information system acquisition methods

b) Explain the criteria for choosing an information system acquisition method

Question 2

a) Identify the emerging trends in Systems Analysis and Design

b) Describe the challenges of emerging trends in Systems Analysis and Design

c) Explain how organizations cope with the challenges of emerging trends in Systems Analysis and Design