M.C.A. SEM –I ,PAPER -II

SYSTEM ANALYSIS AND DESIGN

1. **Introduction**

   - Systems and computer based systems, types of information system
   - System analysis and design
   - Role, task and attribute of the system analyst

2. **Approaches to System development**

   - SDLC
   - Explanation of the phases
   - Different models their advantages and disadvantages
     - o Waterfall approach
     - o Iterative approach
     - o Extreme programming
     - o RAD model
     - o Unified process
     - o Evolutionary software process model
       - Incremental model
       - Spiral model
       - Concurrent development model

3. **Analysis : Investigating System Requirements**

   - Activities of the analysis phase
   - Fact finding methods
     - o Review existing reports, forms and procedure descriptions
     - o Conduct interviews
     - o Observe and document business processes
     - o Build prototypes
     - o Questionnaires
     - o Conduct jad sessions
   - Validate the requirements
     - o Structured walkthroughs

4. **Feasibility Analysis**

- Feasibility Study and Cost Estimates
- Cost benefit analysis
- Identification of list of deliverables

5. **Modeling System Requirements**

- Data flow diagram logical and physical
- Structured English
- Decision tables
- Decision trees
- Entity relationship diagram
- Data dictionary

6. **Design**

- Design phase activities
- Develop System Flowchart
- Structure Chart
  - Transaction Analysis
  - Transform Analysis

Software design and documentation tools

- Hipo chart
- Warnier orr diagram

Designing databases

- Entities
- Relationships
- Attributes
- Normalization

7. **Designing input, output and interface**

- Input design
- Output design
- User interface design

8. **Testing**

- Strategic approach to software testing
- Test series for conventional software
- Test strategies for object – oriented software
- Validation testing
- System testing
- Debugging

9. Implementation and Maintenance

- Activities of the implementation and support phase

10. Documentation

Use of case tools,

Documentation – importance, types of documentation

Books :

1. "Analysis and Design of Information Systems" : Senn, TMH.
2. System Analysis and Design : Howryskiewycz, PHI.
3. "System Analysis and Design" : Awad.
4. "Software Engineering A practitioners Approach" : Roger S. Pressman TMH.
5. "System Analysis and Design Methods : "Whitten, Bentley.
6. "Analysis and Design of Information Systems" : Rajaraman, PHI.

❖❖❖❖

1

INTRODUCTION

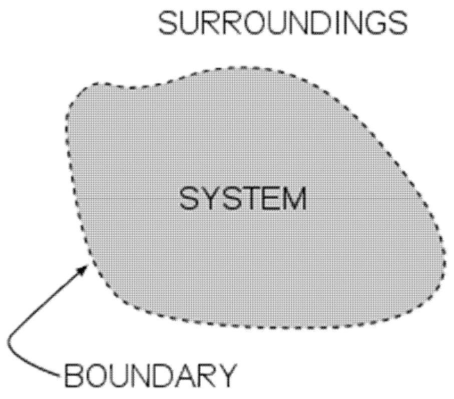
Unit Structure

1.1 INTRODUCTION:

System is combination of different factors which perform different functions. It handles by user and administrator who has a knowledge and skill about that system.

1.2 SYSTEM

The concept of an 'integrated whole' can also be stated in terms of a system embodying a set of relationships which are differentiated from relationships of the set to other elements, and from relationships between an element of the set and elements not a part of the relational regime.



- Systems have structure, defined by parts and their composition;
- Systems have behavior, which involves inputs, processing and outputs of material, energy or information;
- Systems have interconnectivity: the various parts of a system have functional as well as structural relationships between each other.
- Systems have by themselves functions or groups of functions

1.3 CLASSIFICATION OF SYSTEM :

Classification of systems can be done in many ways.

1.3.1 Physical or Abstract System

Physical systems are tangible entities that we can feel and touch. These may be static or dynamic in nature. For example, take a computer center. Desks and chairs are the static parts, which assist in the working of the center. Static parts don't change. The dynamic systems are constantly changing. Computer systems are dynamic system. Programs, data, and applications can change according to the user's needs.

Abstract systems are conceptual. These are not physical entities. They may be formulas, representation or model of a real system.

1.3.2 Open Closed System

Systems interact with their environment to achieve their targets. Things that are not part of the system are environmental elements for the system. Depending upon the interaction with the environment, systems can be divided into two categories, open and closed.

Open systems: Systems that interact with their environment. Practically most of the systems are open systems. An open system has many interfaces with its environment. It can also adapt to changing environmental conditions. It can receive inputs from, and delivers output to the outside of system. An information system is an example of this category.

Closed systems: Systems that don't interact with their environment. Closed systems exist in concept only.

1.3.3 Man made Information System

The main purpose of information systems is to manage data for a particular organization. Maintaining files, producing information and reports are few functions. An information system produces customized information depending upon the needs of the organization. These are usually formal, informal, and computer based.

Formal Information Systems: It deals with the flow of information from top management to lower management. Information flows in the form of memos, instructions, etc. But feedback can be given from lower authorities to top management.

Informal Information systems: Informal systems are employee based. These are made to solve the day to day work related problems. Computer-Based Information Systems: This class of systems depends on the use of computer for managing business applications.

1.3.4 Computer Base System:

A system of one or more computers and associated software with common storage called system.

A computer is a programmable machine that receives input, stores and manipulates data, and provides output in a useful format.

The computer elements described thus far are known as "hardware." A computer system has three parts: the hardware, the software, and the people who make it work.

1.3.5 Information System:

An information system (IS) is any combination of information technology and people's activities using that technology to support operations, management, and decision-making.

Information system deals with data of the organizations. The purposes of Information system are to process input, maintain data, produce reports, handle queries, handle on line transactions, generate reports, and other output. These maintain huge databases, handle hundreds of queries etc. The transformation of data into information is primary function of information system.

Information systems differ in their business needs. Also depending upon different levels in organization information systems differ. Three major information systems are

1. Transaction processing systems
2. Management information systems
3. Decision support systems

Figure 1.2 shows relation of information system to the levels of organization. The information needs are different at different organizational levels. Accordingly the information can be categorized as: strategic information, managerial information and operational information.

Strategic information is the information needed by top most management for decision making. For example the trends in revenues earned by the organization are required by the top management for setting the policies of the organization. This information is not required by the lower levels in the organization. The information systems that provide these kinds of information are known as Decision Support Systems.

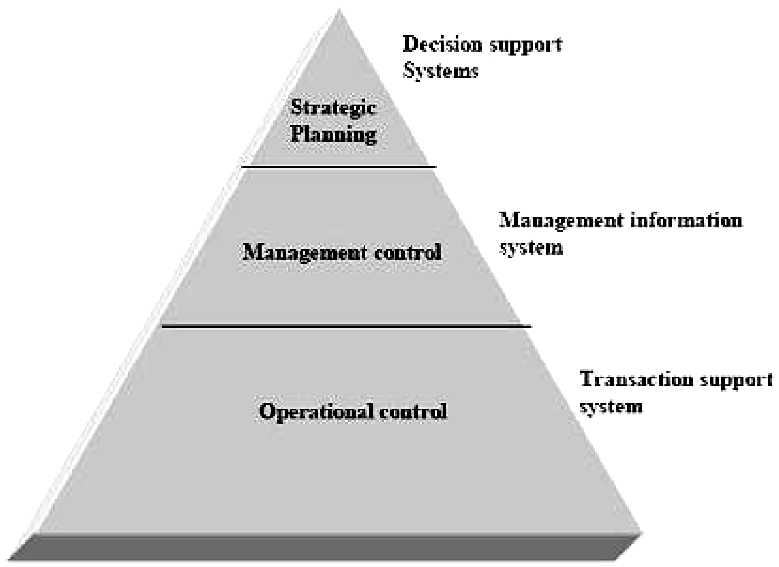


Figure - Relation of information systems to levels of organization

The second category of information required by the middle management is known as managerial information. The information required at this level is used for making short term decisions and plans for the organization. Information like sales analysis for the past quarter or yearly production details etc. fall under this category. Management information system (MIS) caters to such information needs of the organization. Due to its capabilities to fulfill the managerial information needs of the organization, Management Information Systems have become a necessity for all big organizations. And due to its vastness, most of the big organizations have separate MIS departments to look into the related issues and proper functioning of the system.

The third category of information is relating to the daily or short term information needs of the organization such as attendance records of the employees. This kind of information is required at the operational level for carrying out the day-to-day operational activities. Due to its capabilities to provide information for processing transaction of the organization, the information system is known as Transaction Processing System or Data Processing System. Some examples of information provided by such systems are processing of orders, posting of entries in bank, evaluating overdue purchaser orders etc.

1.3.6 Transaction Processing Systems

TPS processes business transaction of the organization. Transaction can be any activity of the organization. Transactions differ from organization to organization. For example, take a railway reservation system. Booking, cancelling, etc are all transactions.

Any query made to it is a transaction. However, there are some transactions, which are common to almost all organizations. Like employee new employee, maintaining their leave status, maintaining employees accounts, etc.

This provides high speed and accurate processing of record keeping of basic operational processes. These include calculation, storage and retrieval.

Transaction processing systems provide speed and accuracy, and can be programmed to follow routines functions of the organization.

1.3.7 Management Information Systems:

These systems assist lower management in problem solving and making decisions. They use the results of transaction processing and some other information also. It is a set of information processing functions. It should handle queries as quickly as they arrive. An important element of MIS is database.

A database is a non-redundant collection of interrelated data items that can be processed through application programs and available to many users.

1.3.8 Decision Support Systems:

These systems assist higher management to make long term decisions. These type of systems handle unstructured or semi structured decisions. A decision is considered unstructured if there are no clear procedures for making the decision and if not all the factors to be considered in the decision can be readily identified in advance.

These are not of recurring nature. Some recur infrequently or occur only once. A decision support system must very flexible. The user should be able to produce customized reports by giving particular data and format specific to particular situations.

1.4 SYSTEM ANALYSIS :

Systems analysis is the study of sets of interacting entities, including computer systems. This field is closely related to operations research. It is also "an explicit formal carried out to help , referred to as the decision maker, identify a better course of action.

Computers are fast becoming our way of life and one cannot imagine life without computers in today’s world. You go to a railway station for reservation, you want to web site a ticket for a cinema, you go to a library, or you go to a bank, you will find computers at all places. Since computers are used in every possible field today, it becomes an important issue to understand and build these computerized systems in an effective way.

1.5 SOFTWARE ENGINEERING:

Software Engineering is the systematic approach to the development, operation and maintenance of software. Software Engineering is concerned with development and maintenance of software products.

Software engineering (SE) is a profession dedicated to designing, implementing, and modifying software so that it is of higher quality, more affordable, maintainable, and faster to build. It is a "systematic approach to the analysis, design, assessment, implementation, test, maintenance and reengineering of software, that is, the application of engineering to software.

The primary goal of software engineering is to provide the quality of software with low cost. Software Engineering involves project planning, project management, systematic analysis, design, validations and maintenance activities.

Every Engineer wants to design the general theme for to develop the software. So, the stepwise execution is necessary to develop a good software. It is called as software engineering.

1.6 SYSTEM DESIGN :

Systems design is the process or art of defining the architecture, components, modules, interfaces, and data for a system to satisfy specified requirements. One could see it as the application of systems theory to product development. There is some overlap with the disciplines of systems analysis, systems architecture and systems engineering.

System design is divided into two types:

1.6.1 Logical Design

The logical design of a system pertains to an abstract representation of the data flows, inputs and outputs of the system. This is often conducted via modeling, which involves a simplistic (and sometimes graphical) representation of an actual system. In the context of systems design, modeling can undertake the following forms, including:

- Data flow diagrams
- Entity Life Histories
- Entity Relationship Diagrams

1.6.2 Physical Design

The physical design relates to the actual input and output processes of the system. This is laid down in terms of how data is inputted into a system, how it is verified/authenticated, how it is processed, and how it is displayed as output.

Physical design, in this context, does not refer to the tangible physical design of an information system. To use an analogy, a personal computer's physical design involves input via a keyboard, processing within the CPU, and output via a monitor, printer, etc. It would not concern the actual layout of the tangible hardware, which for a PC would be a monitor, CPU, motherboard, hard drive, modems, video/graphics cards, USB slots, etc.

System Design includes following points:

- Requirements analysis - analyzes the needs of the end users or customers
- Benchmarking — is an effort to evaluate how current systems are used
- Systems architecture - creates a blueprint for the design with the necessary specifications for the hardware, software, people and data resources. In many cases, multiple architectures are evaluated before one is selected.
- Design — designers will produce one or more 'models' of what they see a system eventually looking like, with ideas from the analysis section either used or discarded. A document will be produced with a description of the system, but nothing is specific — they might say 'touch screen' or 'GUI operating system', but not mention any specific brands;
- Computer programming and debugging in the software world, or detailed design in the consumer, enterprise or commercial world - specifies the final system components.
- System testing - evaluates the system's actual functionality in relation to expected or intended functionality, including all integration aspects.

1.7 SYSTEM ANALYST:

The system analyst is the person (or persons) who guides through the development of an information system. In performing these tasks the analyst must always match the information system objectives with the goals of the organization.

1.7.1 Role of System Analyst:

Role of System Analyst differs from organization to organization. Most common responsibilities of System Analyst are following :

1) System analysis

It includes system's study in order to get facts about business activity. It is about getting information and determining requirements. Here the responsibility includes only requirement determination, not the design of the system.

2) System analysis and design:

Here apart from the analysis work, Analyst is also responsible for the designing of the new system/application.

3) Systems analysis, design, and programming:

Here Analyst is also required to perform as a programmer, where he actually writes the code to implement the design of the proposed application.

Due to the various responsibilities that a system analyst requires to handle, he has to be multifaceted person with varied skills required at various stages of the life cycle. In addition to the technical know-how of the information system development a system analyst should also have the following knowledge.

- Business knowledge: As the analyst might have to develop any kind of a business system, he should be familiar with the general functioning of all kind of businesses.
- Interpersonal skills: Such skills are required at various stages of development process for interacting with the users and extracting the requirements out of them
- Problem solving skills: A system analyst should have enough problem solving skills for defining the alternate solutions to the system and also for the problems occurring at the various stages of the development process.

1.7.2 Task of System Analyst:

The primary objective of any system analyst is to identify the need of the organization by acquiring information by various means and methods. Information acquired by the analyst can be either computer based or manual. Collection of information is the vital step as indirectly all the major decisions taken in the organizations are influenced. The system analyst has to coordinate with the system users, computer programmers, manager and number of people who are related with the use of system. Following are the tasks performed by the system analyst:

Defining Requirement: The basic step for any system analyst is to understand the requirements of the users. This is achieved by various fact finding techniques like interviewing, observation, questionnaire etc. The information should be collected in such a way that it will be useful to develop such a system which can provide additional features to the users apart from the desired.

Prioritizing Requirements: Number of users uses the system in the organization. Each one has a different requirement and retrieves different information. Due to certain limitations in computing capacity it may not be possible to satisfy the needs of all the users. Even if the computer capacity is good enough is it necessary to take some tasks and update the tasks as per the changing requirements. Hence it is important to create list of priorities according to users requirements. The best way to overcome the above limitations is to have a common formal or informal discussion with the users of the system. This helps the system analyst to arrive at a better conclusion.

Gathering Facts, data and opinions of Users: After determining the necessary needs and collecting useful information the analyst starts the development of the system with active cooperation from the users of the system. Time to time, the users update the analyst with the necessary information for developing the system. The analyst while developing the system continuously consults the users and acquires their views and opinions.

Evaluation and Analysis: As the analyst maintains continuous he constantly changes and modifies the system to make it better and more user friendly for the users.

Solving Problems: The analyst must provide alternate solutions to the management and should a in dept study of the system to avoid future problems. The analyst should provide with some flexible alternatives to the management which will help the manager to pick the system which provides the best solution.

Drawing Specifications: The analyst must draw certain specifications which will be useful for the manager. The analyst should lay the specification which can be easily understood by the manager and they should be purely non-technical. The specifications must be in detailed and in well presented form.

1.7.3 Attributes of System Analyst:

A System Analyst (S*A*) analyzes the organization and design of businesses, government departments, and non-profit organizations; they also assess business models and their integration with technology.

There are at least four tiers of business analysis:

1. Planning Strategically - The analysis of the organization business strategic needs
2. Operating/Business model analysis - the definition and analysis of the organization's policies and market business approaches
3. Process definition and design - the business process modeling (often developed through process modeling and design)
4. IT/Technical business analysis - the interpretation of business rules and requirements for technical systems (generally IT)

Within the systems development life cycle domain (SDLC), the business analyst typically performs a liaison function between the business side of an enterprise and the providers of services to the enterprise. A Common alternative role in the IT sector is business analyst, systems analyst, and functional analyst, although some organizations may differentiate between these titles and corresponding responsibilities.

1.7.4 Skill required for System Analyst:

Interpersonal skills are as follows:

1: Communication:

It is an interpersonal quality; the system analyst must have command on English language. Communication is necessary to establish a proper relationship between system analyst and the user.

Communication is need to Gather correct information Establishes a problem solving ideas in front of the management.

2: Understanding:

This is also an interpersonal quality of the system analyst, understanding includes

Understanding of the objectives of the organization.
Understanding the problems of the system.

Understanding the information given by the user or employee of the organization.

3: Selling:

The ideas of the system analyst are his products which he sells to the manager of a particular organization. The system analyst must have not only the ability of creating ideas but also to sell his ideas.

4: Teaching:

It is also an interpersonal quality. A system analyst must have teaching skills. He must have the ability to teach team members and the users. He has to teach about the new system and also about the proper use of the new system.

5: New technology:

An analyst is an agent of change, he or she must have the ability to show all the benefits of the candidate system with the new technological advancement, he must knew about Email Internet Advance graphics Server based networking Network technology etc.

1.8 SUMMARY

This chapter is based on System, their entire factors and impact on surrounding. System is divided into different types and it performs various functions. System analyst can handle every types of system.

Questions:

1. What is system? Explain classification of system?

   Ans: Refer 1.2 and 1.3

2. Explain skill of system analyst?

   Ans: refer 1.7.4

❖❖❖❖

2

APPROACHES TO SYSTEM DEVELOPMENT

Unit Structure

2.1 INTRODUCTION:

SDLC**,** It is System Development Life Cycle. It includes Guidance, policies, and procedures for developing systems throughout their life cycle, including requirements, design, implementation, testing, deployment, operations, and maintenance.

2.2 THE SYSTEMS DEVELOPMENT LIFE CYCLE (SDLC), OR SOFTWARE DEVELOPMENT LIFE CYCLE :

In systems engineering, information systems and software engineering, is the process of creating or altering systems, and the models and methodologies that people use to develop these systems. The concept generally refers to computer or information systems.

Systems and Development Life Cycle (SDLC) is a process used by a systems analyst to develop an information system, including requirements, validation, training, and user (stakeholder) ownership. Any SDLC should result in a high quality system that meets or exceeds customer expectations, reaches completion within time and cost estimates, works effectively and efficiently in the current and planned Information Technology infrastructure, and is inexpensive to maintain and cost-effective to enhance.

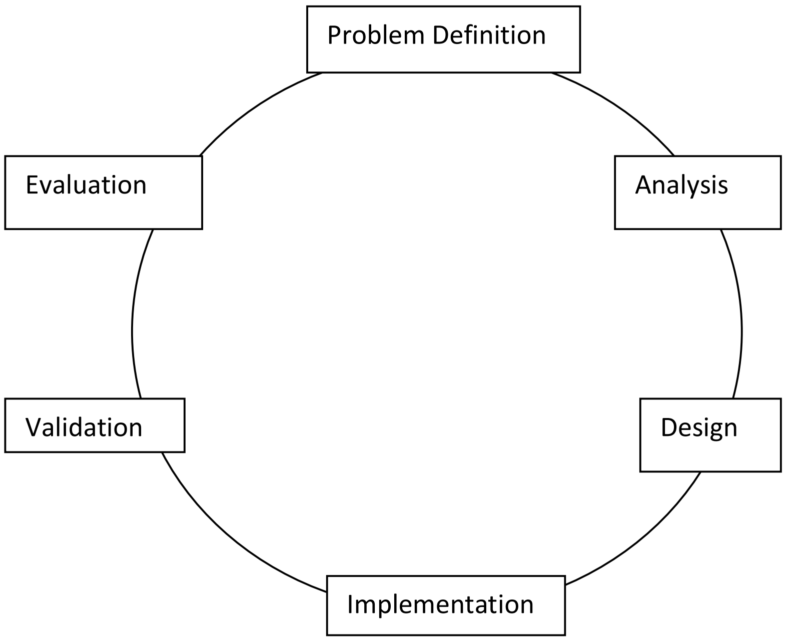
For ex. Computer systems are complex and often (especially with the recent rise of Service-Oriented Architecture) link multiple traditional systems potentially supplied by different software vendors. To manage this level of complexity, a number of SDLC models have been created: "waterfall"; "fountain"; "spiral"; "build and fix"; "rapid prototyping"; "incremental"; and "synchronize and stabilize.

The systems development life cycle (SDLC) is a type of methodology used to describe the process for building information systems, intended to develop information systems in a very deliberate, structured and methodical way, reiterating each stage of the life cycle.

2.2.1 System Development Phases:

Systems Development Life Cycle (SDLC) adheres to important phases that are essential for developers, such as planning, analysis, design, and implementation, and are explained in the section below. Several Systems Development Life Cycle Models exist, the oldest of which — originally regarded as "the Systems Development Life Cycle" — is the waterfall model: a sequence of stages in which the output of each stage becomes the input for the next. These stages generally follow the same basic steps, but many different waterfall methodologies give the steps different names and the number of steps seems to vary between four and seven.

2.2.2 SDLC Phases Diagram:



2.2.3 Explanation of the SDLC Phases:

Requirements gathering and analysis

The goal of system analysis is to determine where the problem is in an attempt to fix the system. This step involves "breaking down" the system in different pieces to analyze the situation, analyzing project goals, "breaking down" what needs to be created and attempting to engage users so that definite requirements can be defined (Decomposition computer science). Requirements Gathering sometimes requires individuals/teams from client as well as service provider sides to get detailed and accurate requirements....

Design:

In systems, design functions and operations are described in detail, including screen layouts, business rules, process diagrams and other documentation. The output of this stage will describe the new system as a collection of modules or subsystems.

The design stage takes as its initial input the requirements identified in the approved requirements document. For each requirement, a set of one or more design elements will be produced as a result of interviews, workshops, and/or prototype efforts. Design elements describe the desired software features in detail, and generally include functional hierarchy diagrams, screen layout diagrams, tables of business rules, business process diagrams, pseudocode, and a complete entity-relationship diagram with a full data dictionary. These design elements are intended to describe the software in sufficient detail that skilled programmers may develop the software with minimal additional input design.

Build or coding:

Modular and subsystem programming code will be accomplished during this stage. Unit testing and module testing are done in this stage by the developers. This stage is intermingled with the next in that individual modules will need testing before integration to the main project.

Testing:

The code is tested at various levels in software testing. Unit, system and user acceptance testings are often performed. This is a grey area as many different opinions exist as to what the stages of testing are and how much if any iteration occurs. Iteration is not generally part of the waterfall model, but usually some occur at this stage.

Below are the following types of testing:

- Data set testing.
- Unit testing
- System testing
- Integration testing
- Black box testing
- White box testing
- Regression testing
- Automation testing
- User acceptance testing
- Performance testing
- Production

definition:- it is a process that ensures that the program performs the intended task.

Operations and maintenance

The deployment of the system includes changes and enhancements before the decommissioning or sunset of the system. Maintaining the system is an important aspect of SDLC. As key personnel change positions in the organization, new changes will be implemented, which will require system updates.

2.2.4 SDLC Phases with Management Control :

The Systems Development Life Cycle (SDLC) phases serve as a programmatic guide to project activity and provide a flexible but consistent way to conduct projects to a depth matching the scope of the project. Each of the SDLC phase objectives are described in this section with key deliverables, a description of recommended tasks, and a summary of related control objectives for effective management. It is critical for the project manager to establish and monitor control objectives during each SDLC phase while executing projects. Control objectives help to provide a clear statement of the desired result or purpose and should be used throughout the entire SDLC process. Control objectives can be grouped into major categories (Domains), and relate to the SDLC phases as shown in the figure.

To manage and control any SDLC initiative, each project will be required to establish some degree of a Work Breakdown Structure(WBS) to capture and schedule the work necessary to complete the project. The WBS and all programmatic material should be kept in the “Project Description” section of the project notebook. The WBS format is mostly left to the project manager to establish in a way that best describes the project work. There are some key areas that must be defined in the WBS as part of the SDLC policy. The following diagram describes three key areas that will be addressed in the WBS in a manner established by the project manager.

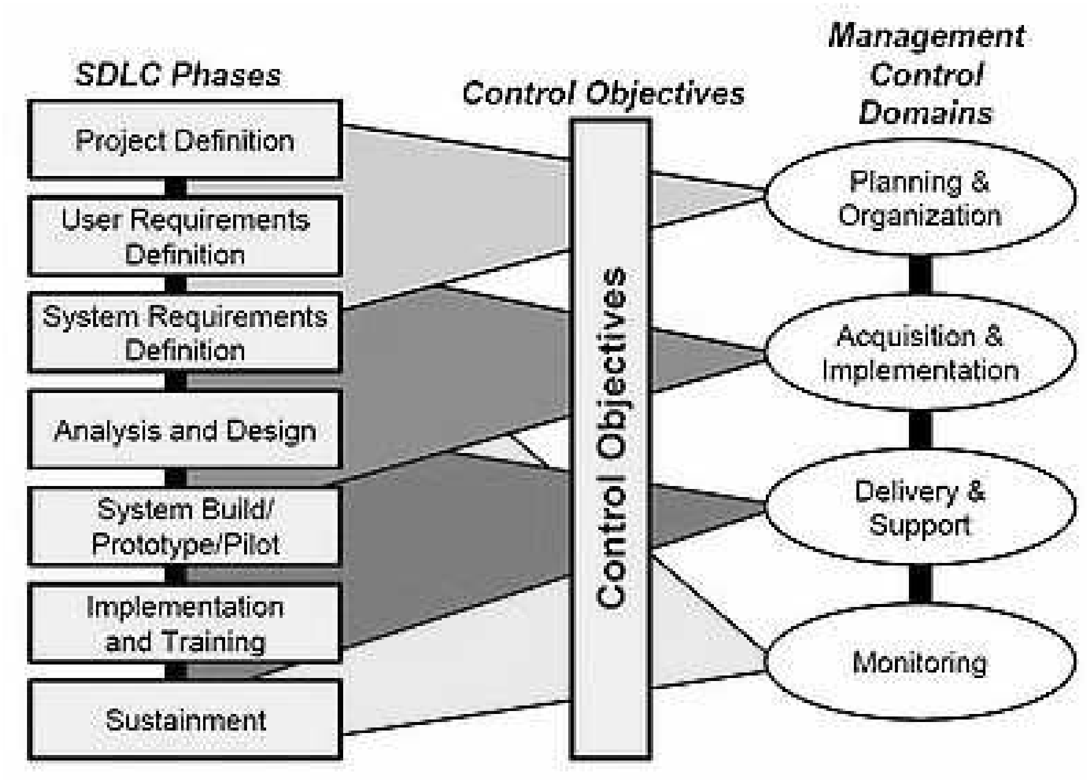

SDLC Phases
Project Definition
User Requirements Definition
System Requirements Definition
Analysis and Design
System Build/ Prototype/Pilot
Implementation and Training
Sustainment
Control Objectives
Control Objectives
Management Control Domains
Planning & Organization
Acquisition & Implementation
Delivery & Support
Monitoring

Diagram: SDLC Phases Related to Management Controls

2.2.5 Advantages of SDLC model:

With an SDLC Model, developers will have a clear idea on what should be or shouldn't be built. Since they already have an idea on the problems that should be answered, a detailed plan could be created following a certain SDLC model. With an SDLC model, developers could even create a program that will answer different problems at the same time. Since everything will be laid out before a single code is written, the goal is clear and could be implemented on time. Although there is a great possibility of deviation from the plan, a good project manager will take care of that concern.

With an SDLC Model, programs built will have a clear documentation of development, structure and even coding. In case there are problems once the program is adopted for public use, developers will always have the documentation to refer to when they need to look for any loopholes. Instead of testing it over and over again which will stop the implementation for a while, developers will just look at the documentation and perform proper maintenance program. This means SDLC will breathe more life to the program. Instead of frustrating developers in uesswork if something goes wrong, SDLC will make sure everything goes smoothly. It will also be a tool for maintenance, ensuring the program created will last for a long time.

2.2.6 Disadvantages of SDLC Model:

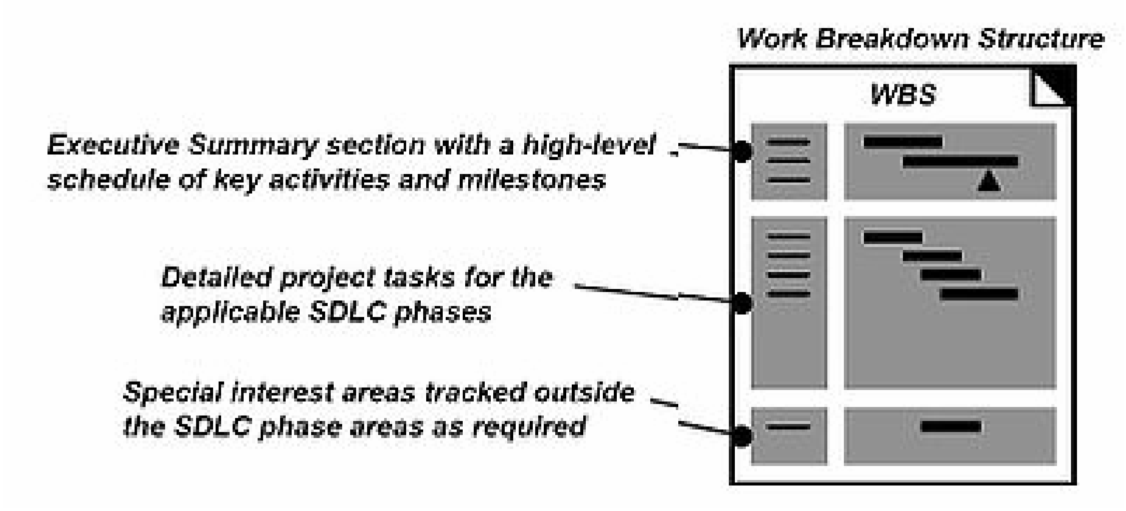
Thinking about the disadvantages of a SDLC model is like looking for a needle in the haystack. But the closest disadvantage anyone could think of SDLC is the difference between what is written in paper and what is actually implemented. There are things that are happening in the actual work that the paper doesn't see. This gives a good impression for the clients especially for 3rd party developers but when the software is actually launched it's on a very bad situation. The actual situation of software development could be covered by fancy paperwork of SDLC.

Another disadvantage of a program or software that follows the SDLC program is it encourages stiff implementation instead of pushing for creativity in different oftware. Although there are SDLC models where programmers could apply their creative juices, it's always in the realm of what is needed instead of freely implementing what the developers think of necessary in the present environment.

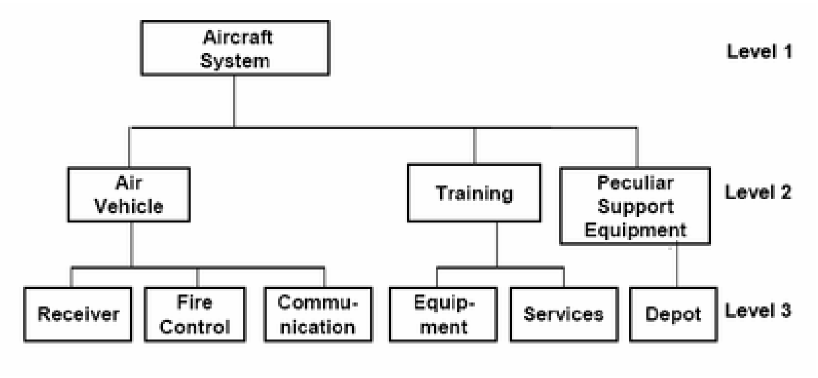
There are so many things that could be done by developers if there are no boundaries or limitations in what should be developed.

2.3 WORK BREAKDOWN STRUCTURE ORGANIZATION:

The upper section of the Work Breakdown Structure (WBS) should identify the major phases and milestones of the project in a summary fashion. In addition, the upper section should provide an overview of the full scope and timeline of the project and will be part of the initial project description effort leading to project approval. The middle section of the WBS is based on the seven Systems Development Life Cycle (SDLC) phases as a guide for WBS task development. The WBS elements should consist of milestones and "tasks" as opposed to "activities" and have a definitive period (usually two weeks or more). Each task must have a measurable output (e.g. document, decision, or analysis). A WBS task may rely on one or more activities (e.g. software engineering, systems engineering) and may require close coordination with other tasks, either internal or external to the project. Any part of the project needing support from contractors should have a Statement of work (SOW) written to include the appropriate tasks from the SDLC phases.


Work Breakdown Structure
WBS
Executive Summary section with a high-level schedule of key activities and milestones
Detailed project tasks for the applicable SDLC phases
Special interest areas tracked outside the SDLC phase areas as required

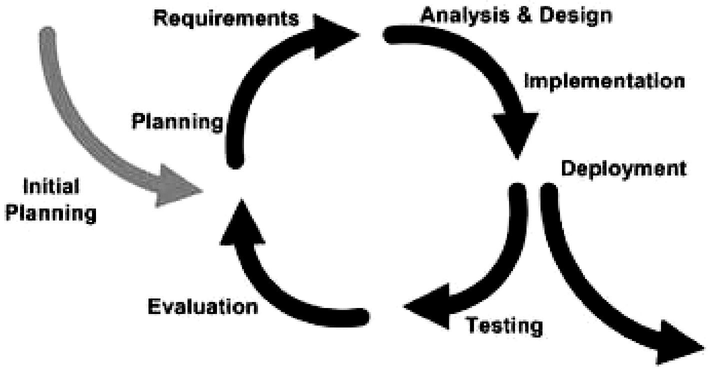
For Ex: Following Diagram indicates Example of a product work breakdown structure of an aircraft system.



2.4 ITERATIVE AND INCREMENTAL DEVELOPMENT MODEL:

Iterative and Incremental development is at the heart of a cyclic software development process developed in response to the weaknesses of the waterfall model. It starts with an initial planning and ends with deployment with the cyclic interactions in between.
Iterative and incremental development is essential parts of the Rational Unified Process, Extreme Programming and generally the various agile software development frameworks.

Diagram : An iterative development model

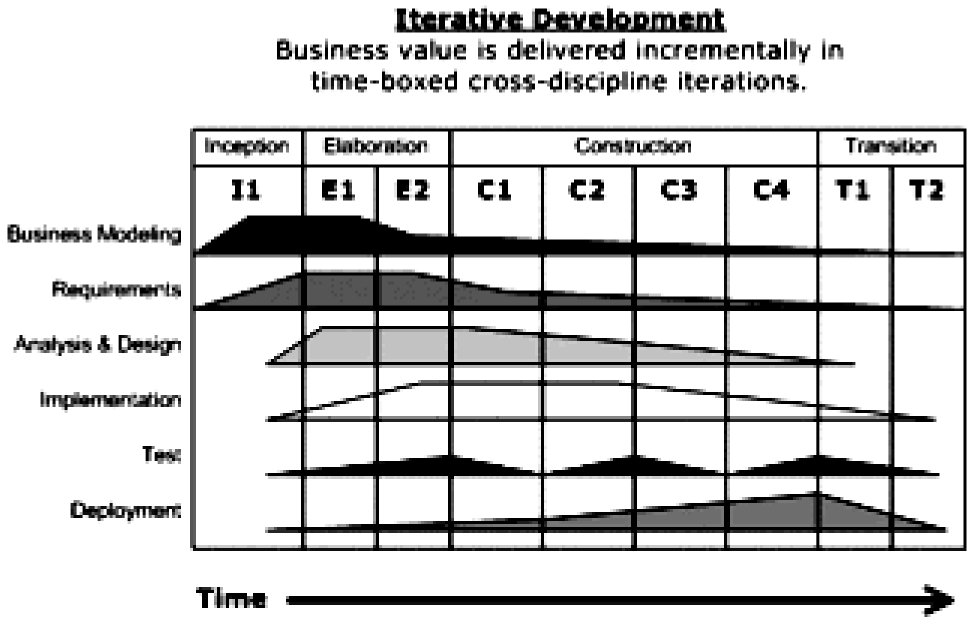


2.4.1 Iterative/Incremental Development

Incremental development slices the system functionality into increments (portions). In each increment, a slice of functionality is delivered through cross-discipline work, from the requirements to the deployment. The unified process groups increments/iterations into phases: inception, elaboration, construction, and transition.

- Inception identifies project scope, risks, and requirements (functional and non-functional) at a high level but in enough detail that work can be estimated.
- Elaboration delivers a working architecture that mitigates the top risks and fulfills the non-functional requirements.
- Construction incrementally fills-in the architecture with production-ready code produced from analysis, design, implementation, and testing of the functional requirements.
- Transition delivers the system into the production operating environment

Diagram: **Iterative/Incremental Development**



2.5 EXTREME PROGRAMMING:

Extreme Programming (XP) is a software development methodology which is intended to improve software quality and responsiveness to changing customer requirements. As a type of agile software development, it advocates frequent "releases" in short development cycles (time boxing), which is intended to improve productivity and introduce checkpoints where new customer requirements can be adopted.

Rules for Extreme Programming:

- Planning
- Managing
- Coding
- Designing
- Testing

2.5.1 Goals of Extreme Programming Model:

Extreme Programming Explained describes Extreme Programming as a software development discipline that organizes people to produce higher quality software more productively.

In traditional system development methods (such as SSADM or the waterfall model) the requirements for the system are determined at the beginning of the development project and often fixed from that point on. This means that the cost of changing the requirements at a later stage (a common feature of software engineering projects) will be high. Like other agile software development methods, XP attempts to reduce the cost of change by having multiple short development cycles, rather than one long one. In this doctrine changes are a natural, inescapable and desirable aspect of software development projects, and should be planned for instead of attempting to define a stable set of requirements.

2.6 RAD MODEL:

It is Rapid Application development model. Rapid Application Development (RAD) refers to a type of software development methodology that uses minimal planning in favour of rapid prototyping. The "planning" of software developed using RAD is interleaved with writing the software itself. The lack of extensive pre-planning generally allows software to be written much faster, and makes it easier to change requirements.

Rapid Application Development is a software development methodology that involves techniques like iterative development and software prototyping. According to Whitten (2004), it is a merger of various structured techniques, especially data-driven Information Engineering, with prototyping techniques to accelerate software systems development.

2.6.1 Practical Application of RAD Model:

When organizations adopt rapid development methodologies, care must be taken to avoid role and responsibility confusion and communication breakdown within the development team, and between the team and the client. In addition, especially in cases where the client is absent or not able to participate with authority in the development process, the system analyst should be endowed with this authority on behalf of the client to ensure appropriate prioritisation of non-functional requirements. Furthermore, no increment of the system should be developed without a thorough and formally documented design phase.

2.6.2 Advantages of the RAD methodology:

1. Flexible and adaptable to changes.
2. Prototyping applications give users a tangible description from which to judge whether critical system requirements are being met by the system. Report output can be compared with existing reports. Data entry forms can be reviewed for completeness of all fields, navigation, data access (drop down lists, checkboxes, radio buttons, etc.).

3. RAD generally incorporates short development cycles - users see the RAD product quickly.
4. RAD involves user participation thereby increasing chances of early user community acceptance.
5. RAD realizes an overall reduction in project risk.
6. Pareto's 80 - 20 Rule usually results in reducing the costs to create a custom system

2.6.3 Disadvantages of RAD methodology:

1. Unknown cost of product. As mentioned above, this problem can be alleviated by the customer agreeing to a limited amount of rework in the RAD process.
2. It may be difficult for many important users to commit the time required for success of the RAD process.

2.7 UNIFIED PROCESS MODEL:

The Unified Software Development Process or Unified Process is a popular iterative and incremental software development process framework. The best-known and extensively documented refinement of the Unified Process is the Rational Unified Process (RUP).

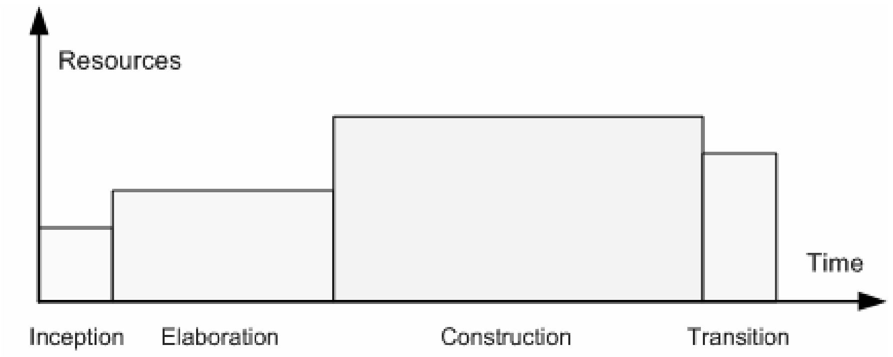


Diagram: Profile of a typical project showing the relative sizes of the four phases of the Unified Process

The Unified Process is not simply a process, but rather an extensible framework which should be customized for specific organizations or projects. The Rational Unified Process is, similarly, a customizable framework. As a result it is often impossible to say whether a refinement of the process was derived from UP or from RUP, and so the names tend to be used interchangeably.

2.7.1 Characteristics:

Iterative and Incremental

The Unified Process is an iterative and incremental development process. The Elaboration, Construction and Transition phases are divided into a series of time boxed iterations. (The Inception phase may also be divided into iterations for a large project.) Each iteration results in an increment, which is a release of the system that contains added or improved functionality compared with the previous release.

Although most iterations will include work in most of the process disciplines (e.g. Requirements, Design, Implementation, Testing) the relative effort and emphasis will change over the course of the project.

2.8 USE CASE DRIVEN

In the Unified Process, use cases are used to capture the functional requirements and to define the contents of the iterations. Each iteration takes a set of use cases or scenarios from requirements all the way through implementation, test and deployment.

2.8.1 Architecture Centric:

The Unified Process insists that architecture sit at the heart of the project team's efforts to shape the system. Since no single model is sufficient to cover all aspects of a system, the Unified Process supports multiple architectural models and views.

One of the most important deliverables of the process is the executable architecture baseline which is created during the Elaboration phase. This partial implementation of the system serves to validate the architecture and act as a foundation for remaining development.

2.8.2 Risk Focused:

The Unified Process requires the project team to focus on addressing the most critical risks early in the project life cycle. The deliverables of each iteration, especially in the Elaboration phase, must be selected in order to ensure that the greatest risks are addressed first.

The Unified Process divides the project into four phases:

- Inception
- Elaboration
- Construction
- Transition

2.8.3 Inception Phase:

Inception is the smallest phase in the project, and ideally it should be quite short. If the Inception Phase is long then it may be an indication of excessive up-front specification, which is contrary to the spirit of the Unified Process.

The following are typical goals for the Inception phase.

- Establish a justification or business case for the project
- Establish the project scope and boundary conditions
- Outline the use cases and key requirements that will drive the design tradeoffs
- Outline one or more candidate architectures
- Identify risks
- Prepare a preliminary project schedule and cost estimate

The Lifecycle Objective Milestone marks the end of the Inception phase.

2.8.4 Elaboration Phase:

During the Elaboration phase the project team is expected to capture a healthy majority of the system requirements. However, the primary goals of Elaboration are to address known risk factors and to establish and validate the system architecture. Common processes undertaken in this phase include the creation of use case diagrams, conceptual diagrams (class diagrams with only basic notation) and package diagrams (architectural diagrams).

The architecture is validated primarily through the implementation of an Executable Architecture Baseline. This is a partial implementation of the system which includes the core, most architecturally significant, components. It is built in a series of small, timeboxed iterations. By the end of the Elaboration phase the system architecture must have stabilized and the executable architecture baseline must demonstrate that the architecture will support the key system functionality and exhibit the right behavior in terms of performance, scalability and cost.

The final Elaboration phase deliverable is a plan (including cost and schedule estimates) for the Construction phase. At this point the plan should be accurate and credible, since it should be based on the Elaboration phase experience and since significant risk factors should have been addressed during the Elaboration phase.

The Lifecycle Architecture Milestone marks the end of the Elaboration phase.

2.8.5 Construction Phase:

Construction is the largest phase in the project. In this phase the remainder of the system is built on the foundation laid in Elaboration. System features are implemented in a series of short, time boxed iterations. Each iteration results in an executable release of the software. It is customary to write full text use cases during the construction phase and each one becomes the start of a new iteration. Common UML (Unified Modelling Language) diagrams used during this phase include Activity, Sequence, Collaboration, State (Transition) and Interaction Overview diagrams.

The Initial Operational Capability Milestone marks the end of the Construction phase.

2.8.6 Transition Phase

The final project phase is Transition. In this phase the system is deployed to the target users. Feedback received from an initial release (or initial releases) may result in further refinements to be incorporated over the course of several Transition phase iterations. The Transition phase also includes system conversions and user training.

2.9 EVOLUTIONARY SOFTWARE PROCESS MODEL:

Software Products can be perceived as evolving over a time period. However, neither the Linear Sequential Model nor the Prototype Model applies this aspect to software production. The Linear Sequential Model was designed for straight-line development. The Prototype Model was designed to assist the customer in understanding requirements and is designed to produce a visualization of the final system.

But the Evolutionary Models take the concept of “evolution” into the engineering paradigm. Therefore Evolutionary Models are **iterative.** They are built in a manner that enables software engineers to develop increasingly more complex versions of the software.

2.9.1 The Incremental Model:

The Incremental Model combines elements of the Linear Sequential Model (applied repetitively) with the iterative philosophy of prototyping. When an Incremental Model is used, the first increment is often the “core product”. The subsequent iterations are the supporting functionalities or the add-on features that a customer would like to see. More specifically, the model is designed, implemented and tested as a series of incremental builds until the product is finished.

2.10 THE SPIRAL MODEL:

The Spiral Model is an evolutionary software process model that couples the iterative nature of prototyping with the controlled and systematic aspects of the Linear Sequential Model. Using the Spiral Model the software is developed in a series of incremental releases. Unlike the Iteration Model where in the first product is a core product, in the Spiral Model the early iterations could result in a paper model or a prototype. However, during later iterations more complex functionalities could be added.

A Spiral Model, combines the iterative nature of prototyping with the controlled and systematic aspects of the Waterfall Model, therein providing the potential for rapid development of incremental versions of the software. A Spiral Model is divided into a number of framework activities, also called task regions. These task regions could vary from 3-6 in number and they are:

- **Customer Communication** - tasks required to establish effective communication between the developer and customer.
- **Planning** - tasks required defining resources, timelines and other project related information /items.
- **Risk Analysis** - tasks required to assess the technical and management risks.
- **Engineering** - tasks required to build one or more representation of the application.
- **Construction & Release** - tasks required to construct, test and support (eg. Documentation and training)
- **Customer evaluation** - tasks required to obtain periodic customer feedback so that there are no last minute surprises.

2.10.1 Advantages of the Spiral Model:

- Realistic approach to the development because the software evolves as the process progresses. In addition, the developer and the client better understand and react to risks at each evolutionary level.
- The model uses prototyping as a risk reduction mechanism and allows for the development of prototypes at any stage of the evolutionary development.
- It maintains a systematic stepwise approach, like the classic waterfall model, and also incorporates into it an iterative framework that more reflect the real world.

2.10.2 Disadvantages of the Spiral Model:

- One should possess considerable risk-assessment expertise
- It has not been employed as much proven models (e.g. the Waterfall Model) and hence may prove difficult to 'sell' to the client.

2.11 CONCURRENT DEVELOPMENT MODEL:

The concurrent development model, sometimes called concurrent engineering.

The concurrent process model can be represented schematically as a series of major technical activities, tasks, and their associated states. For example, the engineering activity defined for the spiral model is accomplished by invoking the following tasks: prototyping and/or analysis modeling, requirements specification, and design.

The activity-analysis-may be in any one of the states noted at any given time. Similarly, other activities (e.g., design or customer communication) can be represented in an analogous manner. All activities exist concurrently but reside in different states. For example, early in a project the customer communication activity has completed its first iteration and exists in the awaiting changes state. The analysis activity (which existed in the none state while initial customer communication was completed) now makes a transition into the under development state. If, however, the customer indicates that changes in requirements must be made, the analysis activity moves from the under development state into the awaiting changes state.

2.12 SUMMARY:

This chapter concern with system and their development models. The system follows their different approaches with the help of different types Model.

Questions:

1. Explain SDLC in detail?

   Ans: refer 2.2.4

2. Explain incremental and iterative model in detail?

   Ans: refer 2.4

❖❖❖❖

3

ANALYSIS: INVESTIGATING SYSTEM REQUIREMENTS

Unit Structure

3.1 INTRODUCTION:

System is concerned with various factors. System require internal and external information/data for to processing functions.

3.2 SYSTEM ANALYSIS:

In this phase, the current system is studied in detail. A person responsible for the analysis of the system is known as analyst. In system analysis, the analyst conducts the following activities.

3.2.1 Needs System Analysis:

This activity is known as requirements analysis. In this step the analyst sums up the requirements of the system from the user and the managers. The developed system should satisfy these requirements during testing phase.

3.2.2 Data Gathering:

In this step, the system analyst collects data about the system to be developed. He uses different tools and methods, depending on situation. These are:

3.2.3 Written Documents:

The analyst may collect the information/data from written documents available from manual-files of an organization. This method of data gathering is normally used if you want to computerize the existing manual system or upgrade the existing computer based system. The written documents may be reports, forms, memos, business plans, policy statements, organizational charts and many others. The written documents provide valuable information about the existing system.

3.2.4 Interviews:

Interview is another data gathering technique. The analyst (or project team members) interviews, managers, users/ clients, suppliers, and competitors to collect the information about the system. It must be noted that the questions to be asked from them should be precise, relevant and to the point.

3.2.5 Questionnaires:

Questionnaires are the feedback forms used to collect Information. The interview technique to collect information is time-consuming method, so Questionnaires are designed to collect information from as many people as we like. It is very convenient and inexpensive method to collect information but sometimes the response may be Confusing or unclear and insufficient.

3.2.6 Observations:

In addition to the above-mentioned three techniques to collect information, the analyst (or his team) may collect Information through observation. In this collect technique, the working, behavior, and other related information of the existing system are observed. It means that working of existing system is watched carefully. 3.2.7 Sampling

If there are large numbers of people or events involved in The system, we can use sampling method to collect information. In this method, only a part of the people or events involved are used to collect information. For example to test the quality of a fruit, we test a piece of the fruit.

3.3 DATA ANALYSIS

After completion of gathering step the collected data about the system is analyzed to ensure that the data is accurate and complete. For this purpose, various tools may be used. The most popular and commonly used tools for data analysis are:

- DFDs (Data Flow Diagrams)
- System Flowcharts
- Connectivity Diagrams
- Grid Charts
- Decision Tables etc.

3.3.1 Analysis Report:

After completing the work of analysis, the requirements collected for the system are documented in a presentable form. It means that the analysis report is prepared. It is done for review and approval of the project from the higher management. This report should have three parts.

- **First**, it should explain how the current system works.
- **Second**, it should explain the problems in the existing system.
- **Finally**, it should describe the requirements for the new system and make recommendations for future.

3.4 FACT FINDING METHODS:

To study any system the analyst needs to do collect facts and all relevant information. the facts when expressed in quantitative form are termed as data. The success of any project is depended upon the accuracy of available data. Accurate information can be collected with help of certain methods/ techniques. These specific methods for finding information of the system are termed as fact finding techniques. Interview, Questionnaire, Record View and Observations are the different fact finding techniques used by the analyst. The analyst may use more than one technique for investigation.

3.4.1 Interview

This method is used to collect the information from groups or individuals. Analyst selects the people who are related with the system for the interview. In this method the analyst sits face to face with the people and records their responses. The interviewer must plan in advance the type of questions he/ she is going to ask and should be ready to answer any type of question. He should also choose a suitable place and time which will be comfortable for the respondent.

The information collected is quite accurate and reliable as the interviewer can clear and cross check the doubts there itself. This method also helps gap the areas of misunderstandings and help to discuss about the future problems. Structured and unstructured are the two sub categories of Interview. Structured interview is more formal interview where fixed questions are asked and specific information is collected whereas unstructured interview is more or less like a casual conversation where in-depth areas topics are covered and other information apart from the topic may also be obtained.

3.4.2 Questionnaire:

It is the technique used to extract information from number of people. This method can be adopted and used only by an skillful analyst. The Questionnaire consists of series of questions framed together in logical manner. The questions are simple, clear and to the point. This method is very useful for attaining information from people who are concerned with the usage of the system and who are living in different countries. The questionnaire can be mailed or send to people by post. This is the cheapest source of fact finding.

3.4.3 Record View

The information related to the system is published in the sources like newspapers, magazines, journals, documents etc. This record review helps the analyst to get valuable information about the system and the organization.

3.4.4 Observation

Unlike the other fact finding techniques, in this method the analyst himself visits the organization and observes and understand the flow of documents, working of the existing system, the users of the system etc. For this method to be adopted it takes an analyst to perform this job as he knows which points should be noticed and highlighted. In analyst may observe the unwanted things as well and simply cause delay in the development of the new system.

3.5 CONDUCT INTERVIEWS:

Interviews are particularly useful for getting the story behind a participant's experiences. The interviewer can pursue in-depth information around a topic. Interviews may be useful as follow-up to certain respondents to questionnaires, e.g., to further investigate their responses. Usually open-ended questions are asked during interviews.

Before you start to design your interview questions and process, clearly articulate to yourself what problem or need is to be addressed using the information to be gathered by the interviews. This helps you keep clear focus on the intent of each question.

3.5.1 Preparation for Interview:

1. Choose a setting with little distraction. Avoid loud lights or noises, ensure the interviewee is comfortable (you might ask them if they are), etc. Often, they may feel more comfortable at their own places of work or homes.

2. Explain the purpose of the interview.
3. Address terms of confidentiality. Note any terms of confidentiality. (Be careful here. Rarely can you absolutely promise anything. Courts may get access to information, in certain circumstances.) Explain who will get access to their answers and how their answers will be analyzed. If their comments are to be used as quotes, get their written permission to do so. See getting informed consent.
4. Explain the format of the interview. Explain the type of interview you are conducting and its nature. If you want them to ask questions, specify if they're to do so as they have them or wait until the end of the interview.
5. Indicate how long the interview usually takes.
6. Tell them how to get in touch with you later if they want to.
7. Ask them if they have any questions before you both get started with the interview.
8. Don't count on your memory to recall their answers. Ask for permission to record the interview or bring along someone to take notes.

3.5.2 Types of Interviews:

1. Informal, conversational interview - No predetermined questions are asked, in order to remain as open and adaptable as possible to the interviewee's nature and priorities; during the interview, the interviewer "goes with the flow".
2. General interview guide approach - the guide approach is intended to ensure that the same general areas of information are collected from each interviewee; this provides more focus than the conversational approach, but still allows a degree of freedom and adaptability in getting information from the interviewee.
3. Standardized, open-ended interview - here, the same open-ended questions are asked to all interviewees (an open-ended question is where respondents are free to choose how to answer the question, i.e., they don't select "yes" or "no" or provide a numeric rating, etc.); this approach facilitates faster interviews that can be more easily analyzed and compared.
4. Closed, fixed-response interview - where all interviewees are asked the same questions and asked to choose answers from among the same set of alternatives. This format is useful for those not practiced in interviewing.

3.5.3 Types of Topics in Questions:

1. Behaviors - about what a person has done or is doing
2. Opinions/values - about what a person thinks about a topic
3. Feelings - note that respondents sometimes respond with "I think ..." so be careful to note that you're looking for feelings
4. Knowledge - to get facts about a topic
5. Sensory - about what people have seen, touched, heard, tasted or smelled
6. Background/demographics - standard background questions, such as age, education, etc.

3.5.4 Wording of Questions:

1. Wording should be open-ended. Respondents should be able to choose their own terms when answering questions.
2. Questions should be as neutral as possible. Avoid wording that might influence answers, e.g., evocative, judgmental wording.
3. Questions should be asked one at a time.
4. Questions should be worded clearly. This includes knowing any terms particular to the program or the respondents' culture.
5. Be careful asking "why" questions. This type of question infers a cause-effect relationship that may not truly exist. These questions may also cause respondents to feel defensive, e.g., that they have to justify their response, which may inhibit their responses to this and future questions

3.6 OBSERVE & DOCUMENT BUSINESS PROCESSES:

Business analysis is the discipline of identifying business needs and determining solutions to business problems. Solutions often include a systems development component, but may also consist of process improvement or organizational change or strategic planning and policy development. The person who carries out this task is called a business analyst or BA.

Business analysis as a discipline has a heavy overlap with requirements analysis sometimes also called requirements engineering, but focuses on identifying the changes to an organization that are required for it to achieve strategic goals. These changes include changes to strategies, structures, policies, processes, and information systems.

3.6.1 Examples of business analysis include:

Enterprise analysis or company analysis

focuses on understanding the needs of the business as a whole, its strategic direction, and identifying initiatives that will allow a business to meet those strategic goals.

Requirements planning and management

involves planning the requirements development process, determining which requirements are the highest priority for implementation, and managing change.

Requirements elicitation

describes techniques for collecting requirements from stakeholders in a project.

Requirements analysis

describes how to develop and specify requirements in enough detail to allow them to be successfully implemented by a project team.

Requirements communication

describes techniques for ensuring that stakeholders have a shared understanding of the requirements and how they will be implemented.

Solution assessment and validation

describes how the business analyst can verify the correctness of a proposed solution.

3.7 ROLES OF BUSINESS ANALYSTS:

As the scope of business analysis is very wide, there has been a tendency for business analysts to specialize in one of the three sets of activities which constitute the scope of business analysis.

Strategist

Organizations need to focus on strategic matters on a more or less continuous basis in the modern business world. Business analysts, serving this need, are well-versed in analyzing the strategic profile of the organization and its environment, advising senior management on suitable policies, and the effects of policy decisions.

Architect

Organizations may need to introduce change to solve business problems which may have been identified by the strategic analysis, referred to above. Business analysts contribute by analyzing objectives, processes and resources, and suggesting ways by which re-design

Systems analyst

There is the need to align IT Development with the systems actually running in production for the Business. A long-standing problem in business is how to get the best return from IT investments, which are generally very expensive and of critical, often strategic, importance. IT departments, aware of the problem, often create a business analyst role to better understand, and define the requirements for their IT systems. Although there may be some overlap with the developer and testing roles, the focus is always on the IT part of the change process, and generally, this type of business analyst gets involved, only when a case for change has already been made and decided upon.

3.7.1 Business process improvement:

A business process improvement (BPI) typically involves six steps

1. Selection of process teams and leader

Process teams, comprising 2-4 employees from various departments that are involved in the particular process, are set up. Each team selects a process team leader, typically the person who is responsible for running the respective process.

2. Process analysis training

The selected process team members are trained in process analysis and documentation techniques.

3. Process analysis interview

The members of the process teams conduct several interviews with people working along the processes. During the interview, they gather information about process structure, as well as process performance data.

4. Process documentation

The interview results are used to draw a first process map. Previously existing process descriptions are reviewed and integrated, wherever possible. Possible process improvements, discussed during the interview, are integrated into the process maps.

5. Review cycle

The draft documentation is then reviewed by the employees working in the process. Additional review cycles may be necessary in order to achieve a common view (mental image) of the process with all concerned employees. This stage is an iterative process.

6. Problem analysis

A thorough analysis of process problems can then be conducted, based on the process map, and information gathered about the process. At this time of the project, process goal information from the strategy audit is available as well, and is used to derive measures for process improvement.

3.7.2 Goal of business analysts:

Business analysts want to achieve the following outcomes:

- Reduce waste
- Create solutions
- Complete projects on time
- Improve efficiency
- Document the right requirements

3.8 BUILD PROTOTYPES :

Software prototyping, an activity during certain software development, is the creation of prototypes, i.e., incomplete versions of the software program being developed.

A prototype typically simulates only a few aspects of the features of the eventual program, and may be completely different from the eventual implementation.

The **conventional** purpose of a prototype is to allow users of the software to evaluate developers' proposals for the design of the eventual product by actually trying them out, rather than having to interpret and evaluate the design based on descriptions. Prototyping can also be used by end users to describe and prove requirements that developers have not considered, so "controlling the prototype" can be a key factor in the commercial relationship between developers and their clients.

3.8.1 Prototyping Process:

The process of prototyping involves the following steps

1. **Identify basic requirements**

Determine basic requirements including the input and output information desired. Details, such as security, can typically be ignored.

2. **Develop Initial Prototype**
   The initial prototype is developed that includes only user interfaces Review The customers, including end-users, examine the prototype and provide feedback on additions or changes.

3. **Revise and Enhance the Prototype**
   Using the feedback both the specifications and the prototype can be improved. Negotiation about what is within the scope of the contract/product may be necessary.

3.8.2 Advantages of prototyping:

There are many advantages to using prototyping in software development – some tangible, some abstract.

Reduced time and costs: Prototyping can improve the quality of requirements and specifications provided to developers. Because changes cost exponentially more to implement as they are detected later in development, the early determination of *what the user really wants* can result in faster and less expensive software.

Improved and increased user involvement: Prototyping requires user involvement and allows them to see and interact with a prototype allowing them to provide better and more complete feedback and specifications. The presence of the prototype being examined by the user prevents many misunderstandings and miscommunications that occur when each side believe the other understands what they said. Since users know the problem domain better than anyone on the development team does, increased interaction can result in final product that has greater tangible and intangible quality. The final product is more likely to satisfy the users desire for look, feel and performance.

3.8.3 Disadvantages of prototyping:

Insufficient analysis: The focus on a limited prototype can distract developers from properly analyzing the complete project. This can lead to overlooking better solutions, preparation of incomplete specifications or the conversion of limited prototypes into poorly engineered final projects that are hard to maintain. Further, since a prototype is limited in functionality it may not scale well if the prototype is used as the basis of a final deliverable, which may not be noticed if developers are too focused on building a prototype as a model.

User confusion of prototype and finished system: Users can begin to think that a prototype, intended to be thrown away, is actually a final system that merely needs to be finished or polished. (They are, for example, often unaware of the effort needed to add error-checking and security features which a prototype may not have.) This can lead them to expect the prototype to accurately model the performance of the final system when this is not the intent of the developers. Users can also become attached to features that were included in a prototype for consideration and then removed from the specification for a final system. If users are able to require all proposed features be included in the final system this can lead to conflict.

Developer misunderstanding of user objectives: Developers may assume that users share their objectives (e.g. to deliver core functionality on time and within budget), without understanding wider commercial issues. For example, user representatives attending Enterprise software (e.g. PeopleSoft) events may have seen demonstrations of "transaction auditing" (where changes are logged and displayed in a difference grid view) without being told that this feature demands additional coding and often requires more hardware to handle extra database accesses. Users might believe they can demand auditing on every field, whereas developers might think this is feature creep because they have made assumptions about the extent of user requirements. If the developer has committed delivery before the user requirements were reviewed, developers are between a rock and a hard place, particularly if user management derives some advantage from their failure to implement requirements.

Developer attachment to prototype: Developers can also become attached to prototypes they have spent a great deal of effort producing; this can lead to problems like attempting to convert a limited prototype into a final system when it does not have an appropriate underlying architecture. (This may suggest that throwaway prototyping, rather than evolutionary prototyping, should be used.)

Excessive development time of the prototype: A key property to prototyping is the fact that it is supposed to be done quickly. If the developers lose sight of this fact, they very well may try to develop a prototype that is too complex. When the prototype is thrown away the precisely developed requirements that it provides may not yield a sufficient increase in productivity to make up for the time spent developing the prototype. Users can become stuck in debates over details of the prototype, holding up the development team and delaying the final product.

Expense of implementing prototyping: the start up costs for building a development team focused on prototyping may be high. Many companies have development methodologies in place, and changing them can mean retraining, retooling, or both. Many companies tend to just jump into the prototyping without bothering to retrain their workers as much as they should.

3.9 QUESTIONAIRE:

A **questionnaire** is a research instrument consisting of a series of questions and other prompts for the purpose of gathering information from respondents. Although they are often designed for statistical analysis of the responses, this is not always the case. The questionnaire was invented by Sir Francis Galton.

Questionnaires have advantages over some other types of surveys in that they are cheap, do not require as much effort from the questioner as verbal or telephone surveys, and often have standardized answers that make it simple to compile data. However, such standardized answers may frustrate users. Questionnaires are also sharply limited by the fact that respondents must be able to read the questions and respond to them. Thus, for some demographic groups conducting a survey by questionnaire may not be practical.

3.9.1 Question Construction:

Question types

Usually, a questionnaire consists of a number of questions that the respondent has to answer in a set format. A distinction is made between open-ended and closed-ended questions. An open-ended question asks the respondent to formulate his own answer, whereas a closed-ended question has the respondent pick an answer from a given number of options. The response options for a closed-ended question should be exhaustive and mutually exclusive. Four types of response scales for closed-ended questions are distinguished:

- Dichotomous, where the respondent has two options
- Nominal-polytomous, where the respondent has more than two unordered options
- Ordinal-polytomous, where the respondent has more than two ordered options
- (Bounded)Continuous, where the respondent is presented with a continuous scale

3.9.2 Basic rules for questionnaire item construction

- Use statements which are interpreted in the same way by members of different subpopulations of the population of interest.

- Use statements where persons that have different opinions or traits will give different answers.
- Think of having an "open" answer category after a list of possible answers.
- Use only one aspect of the construct you are interested in per item.
- Use positive statements and avoid negatives or double negatives.
- Do not make assumptions about the respondent.
- Use clear and comprehensible wording, easily understandable for all educational levels
- Use correct spelling, grammar and punctuation.
- Avoid items that contain more than one question per item (e.g. Do you like strawberries and potatoes?)

3.9.3 Questionnaire administration modes:

Main modes of questionnaire administration are:

- Face-to-face questionnaire administration, where an interviewer presents the items orally.
- Paper-and-pencil questionnaire administration, where the items are presented on paper.
- Computerized questionnaire administration, where the items are presented on the computer.
- Adaptive computerized questionnaire administration, where a selection of items is presented on the computer, and based on the answers on those items, the computer selects following items optimized for the tester’s estimated ability or trait.

3.10 JAD SESSIONS:

JAD (Joint Application Development) is a methodology that involves the client or end user in the design and development of an application, through a succession of collaborative workshops called JAD sessions. Chuck Morris and Tony Crawford, both of IBM, developed JAD in the late 1970s and began teaching the approach through workshops in 1980.

The JAD approach, in comparison with the more traditional practice, is thought to lead to faster development times and greater client satisfaction, because the client is involved throughout the development process. In comparison, in the traditional approach to systems development, the developer investigates the system requirements and develops an application, with client input consisting of a series of interviews.

Joint Application Design (JAD) is a process used in the prototyping life cycle area of the Dynamic Systems Development Method (DSDM) to collect business requirements while developing new information systems for a company. "The JAD process also includes approaches for enhancing user participation, expediting development, and improving the quality of specifications." It consists of a workshop where “knowledge workers and IT specialists meet, sometimes for several days, to define and review the business requirements for the system” The attendees include high level management officials who will ensure the product provides the needed reports and information at the end. This acts as “a management process which allows Corporate Information Services (IS) departments to work more effectively with users in a shorter time frame.”

Through JAD workshops the knowledge workers and IT specialists are able to resolve any difficulties or differences between the two parties regarding the new information system. The workshop follows a detailed agenda in order to guarantee that all uncertainties between parties are covered and to help prevent any miscommunications. Miscommunications can carry far more serious repercussions if not addressed until later on in the process. (See below for Key Participants and Key Steps to an Effective JAD). In the end, this process will result in a new information system that is feasible and appealing to both the designers and end users.

"Although the JAD design is widely acclaimed, little is actually known about its effectiveness in practice." According to Journal of Systems and Software, a field study was done at three organizations using JAD practices to determine how JAD influenced system development outcomes. The results of the study suggest that organizations realized modest improvement in systems development outcomes by using the JAD method. JAD use was most effective in small, clearly focused projects and less effective in large complex projects.

Joint Application Design (JAD) was developed by Drake and Josh of IBM Raleigh and Tony Crawford of IBM Toronto in a workshop setting. Originally, JAD was designed to bring system developers and users of varying backgrounds and opinions together in a productive as well as creative environment. The meetings were a way of obtaining quality requirements and specifications. The structured approach provides a good alternative to traditional serial interviews by system analysts.

Brain-storming and theory Z principals in JAD: In 1984-5 Moshe Telem of Tel-Aviv University, developed and implemented a JAD conceptual approach that integrates brainstorming and Ouchi's "Japanese Management" theory Z principles for rapid, maximal and attainable requirements analysis through JAD. Telem named his approach Brainstorming a Collective Decision-Making Approach (BCDA) [4]. Telem also developed and implemented a BCDA technique (BCDT)[5], which was successfully used within the setting of a pedagogical management information system project for the Israeli educational system[3]. In this project brainstorming and theory Z principles in JAD proved to be not only feasible but also effective, resulting in a realistic picture of true users' information requirements.

3.10.1 Conduct Jad sessions :

Joint Application Development or JAD as it is commonly known as a process originally developed for designing a computer based system. JAD focuses on the use of highly structured, well planned meetings to identify the key components of system development projects.

The JAD process is based on four simple ideas;

- people who actually do a job have the best understanding of the job
- People who are trained in information technology have the best understanding of the possibilities of that technology.
- Information systems never exist alone
- The best results are obtained when all these groups work together on a project.

The JAD technique is based on the observation that the success of a project can be hampered by poor intra team communication, incomplete requirements definition and lack of consensus. The training teaches the essential skills and techniques need to plan, organize and participate in JAD planning.

AD focuses on the use of highly structured, well planned meetings to identify the key components of system development projects. JAD centers on a structured workshop session. It eliminates many problems with traditional meetings which are like workshops. The sessions are

i) very focused
ii) conducted in a dedicated environment
iii) quickly drive major requirements

The participants include: facilitator, end users, developers, tie breakers, observers and subject matter experts. The success of JAD-based workshop depends on the skill of the facilitators.

3.10.2 Need for to Conduct JAD sessions:

Everybody who is responsible for gathering requirements and developing business systems should attend the JAD training sessions. They are: workshop facilitators, business analysts, system analysts, process analysts, development project leaders, development team members, business managers and Information technology members.

3.10.3 Advantages and Disadvantages

JAD is more expensive and cumbersome, compared to other traditional methods. Many companies find that JAD users participate freely in requirements modeling process. They feel a sense of ownership and support for the new system. One big disadvantage is that it opens up a lot of scope for interpersonal conflict.

Compared with traditional methods, JAD may seem more expensive and can be cumbersome if the group is too large relative to the size of the project. Many companies find, however, that JAD allows key users to participate effectively in the requirements modeling process. When users participate in the systems development process, they are more likely to feel a sense of ownership in the results, and support for the new system. When properly used, JAD can result in a more accurate statement of system requirements, a better understanding of common goals, and a stronger commitment to the success of the new system.

Joint Application Design (JAD) is a process used in the prototyping life cycle area of the Dynamic Systems Development Method (DSDM) to collect business requirements while developing new information systems for a company. "The JAD process also includes approaches for enhancing user participation, expediting development, and improving the quality of specifications." It consists of a workshop where “knowledge workers and IT specialists meet, sometimes for several days, to define and review the business requirements for the system.[1]” The attendees include high level management officials who will ensure the product provides the needed reports and information at the end. This acts as “a management process which allows Corporate Information Services (IS) departments to work more effectively with users in a shorter time frame.

Through JAD workshops the knowledge workers and IT specialists are able to resolve any difficulties or differences between the two parties regarding the new information system. The workshop follows a detailed agenda in order to guarantee that all uncertainties between parties are covered and to help prevent any miscommunications. Miscommunications can carry far more serious repercussions if not addressed until later on in the process. (See below for Key Participants and Key Steps to an Effective JAD). In the end, this process will result in a new information system that is feasible and appealing to both the designers and end users.

"Although the JAD design is widely acclaimed, little is actually known about its effectiveness in practice." According to Journal of Systems and Software, a field study was done at three organizations using JAD practices to determine how JAD influenced system development outcomes. The results of the study suggest that organizations realized modest improvement in systems development outcomes by using the JAD method. JAD use was most effective in small, clearly focused projects and less effective in large complex projects.

3.10.4 Four Principle Steps :

1) **Define session objectives-** The first step for the facilitator together with the project leader is to define the session objectives and answering the questions as to what are the session objectives? What is wanted from the session? Who can help create the deliverables?

2) **Prepare for the session-** The facilitator has primary responsibility for the JAD preparation. Four categories of tasks are involved in preparing for the session.

- Conduct pre-session research
- Create a session agenda
- Arrange session logistics
- Prepare the participants
-

3) **Conduct the JAD session-** The facilitator conducts the JAD session, leading the developers and customers through planned agenda. Conducting the meeting involves:

- Starting and ending time,
-Distributing and following the meeting agenda
-Gaining consensus on the meeting purpose and round rules at the beginning of the meeting
-Keeping the meeting on track.

4) **Procedure the Documents-** It is critical to the success of any JAD session that the information on flip-charts, foils, whiteboard, and discussions be recorded and reviewed by the participants. Each day of the session, the facilitator and scribe should create a draft of the day's results. The final documents from the JAD should be completed as soon as possible after the session. It is primary responsibility of the facilitator and the scribe to:

- organize the final document for easy use by project members
- complete a "Final Draft" document
- distribute it to selected individuals for review
- incorporate revisions as necessary
- distribute the final copy for participant sign-off

JAD improves the final quality of the product by keeping the focus on the upfront of the development cycle thus reducing the errors that are likely to cause huge expenses.

3.11 VALIDATION:

Validation may refer to:

- Validity, in logic, determining whether a statement is true
- Validation and verification, in engineering, confirming that a product or service meets the needs of its users
- Verification and Validation (software), checking that a software system meets specifications and fulfills its intended purpose
- Validation of foreign studies and degrees, processes for transferring educational credentials between countries
- Validation (computer security), the process of determining whether a user or computer program is allowed to do something.
    - Validate (McAfee), a software application for this purpose
- Validation (drug manufacture), documenting that a process or system meets its pre-determined specifications and quality attributes
- Validation (psychology), in psychology and human communication, the reciprocated communication of respect which signifies that the other's opinions are acknowledged, respected and heard

- Data validation, in computer science, ensuring that data inserted into an application satisfies defined formats and other input criteria
- Regression model validation, in statistics, determining whether a model fits the data well

3.12 STRUCTURED WALKTHROUGHS:

In typical project planning, you must define the scope of the work to be accomplished. A typical tool that is used by project managers is the work breakdown structure (WBS). This walkthrough demonstrates a general approach to creating a WBS using Team Foundation Server and Microsoft Project.

This walkthrough is not based on any particular methodology. However, it does use the quality of service requirement and task work item types in the MSF for Agile Software Development process template. The approach used in this walkthrough should be adaptable your own organization's work item types and process.

In this walkthrough, you will complete the following tasks:

- Create a requirement using Team Foundation Server.
- Create tasks using Team Foundation Server.
- Create tasks using Microsoft Project.
- Link tasks and requirements.
- Create a work breakdown structure from tasks in Microsoft Project

The term is often employed in the software industry (see software walkthrough) to describe the process of inspecting algorithms and source code by following paths through the algorithms or code as determined by input conditions and choices made along the way. The purpose of such *code walkthroughs* is generally to provide assurance of the fitness for purpose of the algorithm or code; and occasionally to assess the competence or output of an individual or team.

3.12.1 Types of Walkthroughs

- Specification walkthroughs
  - System specification
  - Project planning
  - Requirements analysis
- Design walkthroughs
  - Preliminary design
  - Design

- Code walkthroughs
- Test walkthroughs
    - Test plan
    - Test procedure

3.12.2 Prerequisites

The following prerequisites must be met to complete this walkthrough.

- Microsoft Project must be installed.
- A team project must be created that uses the MSF for Agile Software Development process template.

3.12.3 Scenario

The scenario for this walkthrough is based on the example Adventure Works team project. Adventure Works is starting a project to set up a Web interface for ordering its products. One of the customer requirements states that customers be able to check on order status after orders are placed. The scope of this work must be defined in a work breakdown structure to a sufficient level of detail to enable project planning to be completed.

The following approach is used by Adventure Works. The project manager must create the WBS and has the help of the team to do this. One person on the team is a database expert and will provide details on what is needed in the database to support the new requirement. She will enter her work details using Team Foundation Server.

The project manager will work with other team members to define additional work to complete the Web interface. Then the project manager will enter those details using Microsoft Project.

Finally, the project manager will create a WBS in Microsoft Visio that can be used in the project planning document.

Throughout this walkthrough you will perform the steps of each role to create the tasks and WBS. When you complete the walkthrough, you will have created the following tasks and subtasks in a Gantt chart and a WBS.

- Order Storage Subsystem
    - Order Tables
    - Order Stored Procedures
- Order Web Interface
    - Order Lookup Web Service
    - Client Order Views

3.13 SUMMARY

Through this chapter we discussed about functionality of system as well as system requirements. The resources to collect the data are an essential of system. Some fact findings methods are an important part here.

Questions:

1. Explain role of business analyst in detail?

   Ans: refer 3.7

2. Explain Data Analysis in detail?

   Ans: refer 3.3

❖❖❖❖

4

FEASIBILITY ANALYSIS

Unit Structure

4.1 INTRODUCTION:

A **feasibility study** is an evaluation of a proposal designed to determine the difficulty in carrying out a designated task. Generally, a feasibility study precedes technical development and project implementation. In other words, a feasibility study is an evaluation or analysis of the potential impact of a proposed project.

4.2 FIVE COMMON FACTORS FOR FEASIBILITY STUDY:

4.2.1 Technology and system feasibility

The assessment is based on an outline design of system requirements in terms of Input, Processes, Output, Fields, Programs, and Procedures. This can be quantified in terms of volumes of data, trends, frequency of updating, etc. in order to estimate whether the new system will perform adequately or not. Technological feasibility is carried out to determine whether the company has the capability, in terms of software, hardware, personnel and expertise, to handle the completion of the project

4.2.2 Economic feasibility:

Economic analysis is the most frequently used method for evaluating the effectiveness of a new system. More commonly known as cost/benefit analysis, the procedure is to determine the benefits and savings that are expected from a candidate system and compare them with costs. If benefits outweigh costs, then the decision is made to design and implement the system. An entrepreneur must accurately weigh the cost versus benefits before taking an action.

Cost-based study: It is important to identify cost and benefit factors, which can be categorized as follows: 1. Development costs; and 2. Operating costs. This is an analysis of the costs to be incurred in the system and the benefits derivable out of the system. Time-based study: This is an analysis of the time required to achieve a return on investments. the benefits derived from the system. The future value of a project is also a factor.

4.2.3 Legal feasibility:

Determines whether the proposed system conflicts with legal requirements, e.g. a data processing system must comply with the local Data Protection Acts.

4.2.4 Operational feasibility:

Operational feasibility is a measure of how well a proposed system solves the problems, and takes advantage of the opportunities identified during scope definition and how it satisfies the requirements identified in the requirements analysis phase of system development.

The Need for Operational Feasibility Studies.

Operational feasibility studies are generally utilized to answer the following questions:

- **Process** – How do the end-users feel about a new process that may be implemented?
- **Evaluation** – Whether or not the process within the organization will work but also if it *can* work.
- **Implementation** – Stakeholder, manager, and end-user tasks.
- **Resistance** – Evaluate management, team, and individual resistance and how that resistance will be handled.
- **In-House Strategies** – How will the work environment be affected? How much will it change?
- **Adapt & Review** – Once change resistance is overcome, explain how the new process will be implemented along with a review process to monitor the process change.

Example:

If an operational feasibility study must answer the six items above, how is it used in the real world? A good example might be if a company has determined that it needs to totally redesign the workspace environment.

After analyzing the technical, economic, and scheduling feasibility studies, next would come the operational analysis. In order to determine if the redesign of the workspace environment would work, an example of an operational feasibility study would follow this path based on six elements:

- **Process** – Input and analysis from everyone the new redesign will affect along with a data matrix on ideas and suggestions from the original plans.
- **Evaluation** – Determinations from the process suggestions; will the redesign benefit everyone? Who is left behind? Who feels threatened?
- **Implementation** – Identify resources both inside and out that will work on the redesign. How will the redesign construction interfere with current work?
- **Resistance** – What areas and individuals will be most resistant**Strategies** – How will the organization deal with the changed workspace environment? Do new processes or structures need to be reviewed or implemented in order for the redesign to be effective?

- **Adapt & Review** – How much time does the organization need to adapt to the new redesign. How will it be reviewed and monitored? What will happen if through a monitoring process, additional changes must be made?

4.2.5 Schedule feasibility:

A project will fail if it takes too long to be completed before it is useful. Typically this means estimating how long the system will take to develop, and if it can be completed in a given time period using some methods like payback period. Schedule feasibility is a measure of how reasonable the project timetable is. Given our technical expertise, are the project deadlines reasonable? Some projects are initiated with specific deadlines. You need to determine whether the deadlines are mandatory or desirable It is an essential type of feasibilty.It makes prototype model with proper time span which allot the steps and their required time duration.

4.3 OTHER FEASIBILITY FACTORS:

4.3.1 Market and real estate feasibility:

Market Feasibility Study typically involves testing geographic locations for a real estate development project, and usually involves parcels of real estate land. Developers often conduct market studies to determine the best location within a jurisdiction, and to test alternative land uses for given parcels. Jurisdictions often require developers to complete feasibility studies before they will approve a permit application for retail, commercial, industrial, manufacturing, housing, office or mixed-use project. Market Feasibility takes into account the importance of the business in the selected area.

4.3.2 Resource feasibility:

This involves questions such as how much time is available to build the new system, when it can be built, whether it interferes with normal business operations, type and amount of resources required, dependencies, etc. Contingency and mitigation plans should also be stated here.

4.3.3 Cultural feasibility

In this stage, the project's alternatives are evaluated for their impact on the local and general culture. For example, environmental factors need to be considered and these factors are to be well known. Further an enterprise's own culture can clash with the results of the project.

4.4 COST ESTIMATES:

4.4.1 Cost/Benefit Analysis

Evaluating Quantitatively Whether to Follow a Course of Action

You may have been intensely creative in generating solutions to a problem, and rigorous in your selection of the best one available. However, this solution may still not be worth implementing, as you may invest a lot of time and money in solving a problem that is not worthy of this effort.

Cost Benefit Analysis or CBA is a relatively* simple and widely used technique for deciding whether to make a change. As its name suggests, you simply add up the value of the benefits of a course of action, and subtract the costs associated with it.

Costs are either one-off, or may be ongoing. Benefits are most often received over time. We build this effect of time into our analysis by calculating a payback period. This is the time it takes for the benefits of a change to repay its costs. Many companies look for payback on projects over a specified period of time.

4.4.2 How to Use the Tool:

In its simple form, cost-benefit analysis is carried out using only financial costs and financial benefits. For example, a simple cost benefit ratio for a road scheme would measure the cost of building the road, and subtract this from the economic benefit of improving transport links. It would not measure either the cost of environmental damage or the benefit of quicker and easier travel to work.

A more sophisticated approach to building a cost benefit models is to try to put a financial value on intangible costs and benefits. This can be highly subjective - is, for example, a historic water meadow worth $25,000, or is it worth $500,000 because if its environmental importance? What is the value of stress-free travel to work in the morning?

These are all questions that people have to answer, and answers that people have to defend.

The version of the cost benefit approach we explain here is necessarily simple. Where large sums of money are involved (for example, in financial market transactions), project evaluation can become an extremely complex and sophisticated art. The fundamentals of this are explained in Principles of Corporate Finance by Richard Brealey and Stewart Myers - this is something of an authority on the subject.

Example:

A sales director is deciding whether to implement a new computer-based contact management and sales processing system. His department has only a few computers, and his salespeople are not computer literate. He is aware that computerized sales forces are able to contact more customers and give a higher quality of reliability and service to those customers. They are more able to meet commitments, and can work more efficiently with fulfilment and delivery staff.

His financial cost/benefit analysis is shown below:

Costs:
New computer equipment:

- 10 network-ready PCs with supporting software @ $2,450 each
- 1 server @ $3,500
- 3 printers @ $1,200 each
- Cabling & Installation @ $4,600
- Sales Support Software @ $15,000

Training costs:

- Computer introduction - 8 people @ $400 each
- Keyboard skills - 8 people @ $400 each
- Sales Support System - 12 people @ $700 each

Other costs:

- Lost time: 40 man days @ $200 / day
- Lost sales through disruption: estimate: $20,000
- Lost sales through inefficiency during first months: estimate: $20,000

Total cost: $114,000

Benefits:

- Tripling of mail shot capacity: estimate: $40,000 / year
- Ability to sustain telesales campaigns: estimate: $20,000 / year
- Improved efficiency and reliability of follow-up: estimate: $50,000 / year
- Improved customer service and retention: estimate: $30,000 / year

- Improved accuracy of customer information: estimate: $10,000 / year
- More ability to manage sales effort: $30,000 / year

Total Benefit: $180,000/year

Payback time: $114,000 / $180,000 = 0.63 of a year = approx. 8 months

4.4.3 Benefits of Cost Estimation :

Cost/Benefit Analysis is a powerful, widely used and relatively easy tool for deciding whether to make a change.

To use the tool, firstly work out how much the change will cost to make. Then calculate the benefit you will from it.

Where costs or benefits are paid or received over time, work out the time it will take for the benefits to repay the costs.

Cost/Benefit Analysis can be carried out using only financial costs and financial benefits. You may, however, decide to include intangible items within the analysis. As you must estimate a value for these, this inevitably brings an element of subjectivity into the process.

Larger projects are evaluated using formal finance/capital budgeting, which takes into account many of the complexities involved with financial Decision Making. This is a complex area and is beyond the scope of this site.

4.5 BUSINESS SYSTEM ANALYSIS:

This process involves interviewing stakeholders and gathering information required to assist us in the development of a web application which closely resembles your business model. In most cases our clients use the following types of services:

Project Roadmap: helps you understand which resouces are required at various stages of the project, assess the overall risks and opportunities, and allocate a realistic timeline and budget for the successful completion and implementation of the project.

Project Blueprint: documents and diagrams each of the various technical and content aspects of the software project and how they fit and flow together, such as the **data model** for the database, **user interface** for the users, and everything in between. This becomes the main guideline for the Programmers, Graphic

Designers, and Content Developers who collaborate with the Project Manager on developing a web application.

4.5.1 Developing a Project Roadmap:

Before starting a web application project we need to have the following information:

Establish the approximate project cost and delivery time Identify the required resources and expertise and where to find them Know the risks involved in each development stage, and plan for how to deal with them early on Gain client agreement on the absolute essentials for each phase of the project. This increases the potential for project success and the long-term return on investment and helps to avoid drastic project delays in the future

As a result of the initial study, the Project Roadmap becomes the foundation of our business agreement with your company. To develop a roadmap, we first communicate with the key stakeholders and study your business model. Project requirements are broken down into manageable stages, with each stage assigned a priority rank and a time and development cost estimate.

Approximately 10 % of a project's time and budget is invested in developing the roadmap. This number could increase if your business concept and model are being implemented for the first time.

4.6 DEVELOPING A BLUEPRINT FOR A WEB APPLICATION:

For larger projects, Programmers, Graphic Designers, and Content developers do not start weaving a web application right away. To increase the likelihood of the project's success, we steer the development process based on the project blueprint prepared by our System Analysts and Information Architects.

The blueprint contains text documents and visual diagrams (ER, UML, IA Garrett, Use Cases, ...). In other words, we will be designing the data model for the database, user interface for the users, and everything in between. We model everything on paper to ensure the development team can achieve their goal of a high quality web application on time and on budget.

These documents are refined and improved as the project moves forward. The project blueprint almost always changes depending on the discoveries and challenges that inevitably arise throughout the development process.

These documents and diagrams become your property once the project is delivered. This allows you to grow and further develop the application in the future. We do not keep any of the information proprietary.

4.6.1 Identification of list of deliverables :

This is related to Project Execution and Control.

List of Deliverables:

Project Execution and Control differs from all other work in that, between the kick-off meeting and project acceptance, all processes and tasks occur concurrently and repeatedly, and continue almost the entire duration of Project Execution and Control.

Thus, the earlier concept of a "process deliverable" is not applicable to Project Execution and Control, and even task deliverables are mostly activities, not products.
Of course, there is the ultimate deliverable – the product of the project.

The following table lists all Project Execution and Control processes, tasks, and their deliverables

4.6.2 Process Descriptions

- 1 Conduct Project Execution and Control Kick-off
- 2 Manage Triple Constraints (Scope, Schedule, Budget)
- 3 Monitor and Control Risks
- 4 Manage Project Execution and Control
- 5 Gain Project Acceptance

1. Conduct Project Execution and Control Kick-Off
Roles

- Project Manager
- Project Sponsor and / or Project Director
- Project Team Members
- Steering Committee
- Stakeholders

Purpose

The purpose of Conduct Project Execution and Control Kick-off is to formally acknowledge the beginning of Project Execution and Control and facilitate the transition from Project Planning. Similar to Project Planning Kick-off, Project Execution and Control Kick-off ensures that the project is still on track and focused on the original business need. Many new team members will be introduced to the project at this point, and must be thoroughly oriented and prepared to begin work. Most importantly, current project status is reviewed and all prior deliverables are re-examined, giving all new team members a common reference point.

Tasks associated with Conduct Project Execution and Control Kick-Off
-Orient New Project Team Members
-Review Project Materials
-Kick Off Project Execution and Control

Orient New Project Team Members

As in Project Planning, the goal of orienting new Project Team members is to enhance their abilities to contribute quickly and positively to the projects desired outcome. If the Project Manager created a Team Member Orientation Packet during Project Planning, the packet should already contain an orientation checklist, orientation meeting agenda, project materials, and logistical information that will again be useful.

The Project Manager should review the contents of the existing Team Member Orientation Packet to ensure that they are current and still applicable to the project. Any changes needed to the contents of the packet should be made at this time. Once updated, packet materials can be photocopied and distributed to new team members to facilitate their orientation process. The Project Manager or Team Leader should conduct one-on-one orientation sessions with new members to ensure that they read and understand the information presented to them.

If the orientation packet was not created during Project Planning and new team members are coming on board, the Project Manager must gather and present information that would be useful to new team members, including:

- All relevant project information from Project Initiation, Project Planning (High Level), and Project Planning (Detail Level).
- Organization charts Project Team, Customer, Performing Organization
- General information on the Customer
- Logistics (parking policy, work hours, building/office security requirements, user id and password, dress code, location of rest rooms, supplies, photocopier, printer, fax, refreshments, etc.)
- Project procedures (team member expectations, how and when to report project time and status, sick time and vacation policy)

Review Project Materials and Current Project Status

Before formally beginning Project Execution and Control, the Project Team should review updated Project Status Reports and the Project Plan. At this point in the project, the Project Plan comprises all deliverables produced during Project Initiation and Project Planning (High Level and Detail):

1. Project Charter, Project Initiation Plan
2. Triple Constraints (Scope, Schedule, Budget)
3. Risk Management Worksheet
4. Description of Stakeholder Involvement
5. Communications Plan
6. Time and Cost Baseline
7. Communications Management Process
8. Change Control Process
9. Acceptance Management Process
10. Issue Management and Escalation Process
11. Training Plan
12. Project Implementation and Transition Plan

Kick off Project Execution and Control

As was the case for Project Initiation and Project Planning, a meeting is conducted to kick off Project Execution and Control. During the meeting, the Project Manager should present the main components of the Project Plan for review. Other items to cover during the meeting include:

- Introduction of new team members
- Roles and responsibilities of each team member
- Restating the objective(s) of the project and goals for Execution and Control
- Latest Project Schedule and timeline
- Project risks and mitigation plans
- Current project status, including open issues and action items

The goal of the kick-off meeting is to verify that all parties involved have consistent levels of understanding and acceptance of the work done so far, to validate expectations pertaining to the deliverables to be produced during Project Execution and Control, and to clarify and gain understanding of the expectations of each team member in producing the deliverables. Attendees at the Project Execution and Control Kick-off Meeting include the Project Manager, Project Team, Project Sponsor and / or Project Director, and any other Stakeholders with a vested interest in the status of the project. This is an opportunity for the Project Sponsor and / or Project Director to reinforce the importance of the project and how it supports the business need.

2. Manage Triple Constraints

Roles

- Project Manager
- Project Sponsor and /or Project Director
- Project Team Member
- Customer Representative
- Steering Committee

Purpose

The Triple Constraints is the term used for a project's inextricably linked constraints: Scope, Schedule, and Budget, with a resulting acceptable Quality. During Project Planning, each section of the Triple Constraints was refined. As project-specific tasks are performed during Project Execution and Control, the Triple Constraint will need to be managed according to the processes established during Project Planning.

The Triple Constraints is not static although Project Planning is complete and has been approved, some components of Triple Constraints will continue to evolve as a result of the execution of project tasks. Throughout the project, as more information about the project becomes known and the product of the project is developed, the Triple Constraints are likely to be affected and will need to be closely managed.

The purpose of the Manage Triple Constraints Task is to:

- Manage Changes to Project Scope
- Control the Project Schedule and Manage Schedule Changes
- Implement Quality Assurance and Quality Control Processes According to the Quality Standards Revised During Project Planning
- Control and Manage Costs Established in the Project Budget

Tasks associated with Manage Triple Constraints

- Manage Project Scope
- Manage Project Schedule
- Implement Quality Control
- Manage Project Budget

Manage Project Scope

During Project Planning, the Project Manager, through regular communication with the Customer Representatives and Project Sponsor and / or Project Director, refined the Project Scope to clearly define the content of the deliverables to be produced during Project Execution and Control. This definition includes a clear description of what will and will not be included in each deliverable.

The process to be used to document changes to the Project Scope was included in the Project Initiation Plan. This process includes a description of the way scope will be managed and how changes to scope will be handled. It is important that the Project Manager enforce this process throughout the entire project, starting very early in Project Execution and Control. Even if a scope change is perceived to be very small, exercising the change process ensures that all parties agree to the change and understand its potential impact. Following the process each and every time scope change occurs will minimize confusion as to what actually constitutes a change. Additionally, instituting the process early will test its effectiveness, get the Customer and Project Sponsor and / or Project Director accustomed to the way change will be managed throughout the remainder of the project, and help them understand their roles as they relate to change.

Manage Project Schedule

During Project Planning (Detail Level), an agreed-upon baseline was established for the Project Schedule. This schedule baseline will be used as a starting point against which performance on the project will be measured. It is one of many tools the Project Manager can use during Project Execution and Control to determine if the project is on track.

Project Team members use the communications mechanisms documented in the Communications Plan to provide feedback to the Project Manager on their progress. Generally team members document the time spent on tasks and provides estimates of the time required to complete them. The Manager uses this information to update the Project Schedule. In some areas there may be formal time tracking systems that are used to track project activity.

After updating the Project Schedule, the Project Manager must take the time to review the status of the project. Some questions that the Project Manager should be able to answer by examining the Project Schedule include:

- Is the project on track?
- Are there any issues that are becoming evident that need to be addressed now?

- Which tasks are taking more time than estimated? Less time?
- If a task is late, what is the effect on subsequent tasks?
- What is the next deliverable to be produced and when is it scheduled to be complete?
- What is the amount of effort expended so far and how much is remaining?
- Are any Project Team members over-allocated or under-allocated?
- How much of the time allocated has been expended to date and what is the time required to complete the project?
- Most project scheduling tools provide the ability to produce reports to display a variety of useful information. It is recommended that the Project Manager experiment with all available reports to find those that are most useful for reporting information to the Project Team, Customer, and Project Sponsor and / or Project Director.
- When updating the Project Schedule, it is very important that the Project Manager maintain the integrity of the current schedule. Each version of the schedule should be archived. By creating a new copy of the schedule whenever it is updated, the Project Manager will never lose the running history of the project and will also have a copy of every schedule for audit purposes.

4.7 IMPLEMENT QUALITY CONTROL

Quality control involves monitoring the project and its progress to determine if the quality standards defined during Project Planning are being implemented and whether the results meet the quality standards defined during Project Initiation. The entire organization has responsibilities relating to quality, but the primary responsibility for ensuring that the project follows its defined quality procedures ultimately belongs to the Project Manager. The following figure highlights the potential results of executing a project with poor quality compared to a project executed with high quality:

Poor Quality	High Quality
Increased costs	Lower costs
Low morale	Happy, productive Project Team
Low Customer satisfaction	Delivery of what the Customer wants
Increased risk	Lower risk

Quality control should be performed throughout the course of the project. Some of the activities and processes that can be used to monitor the quality of deliverables, determine if project results comply with quality standards, and identify ways to improve unsatisfactory performance, are described below. The Project Manager and Project Sponsor and / or Project Director should decide which are best to implement in their specific project environment.

- **Conduct Peer Reviews** the goal of a peer review is to identify and remove quality issues from a deliverable as early in Project Execution and Control as efficiently as possible. A peer review is a thorough review of a specific deliverable, conducted by members of the Project Team who are the day-to-day peers of the individuals who produced the work. The peer review process adds time to the overall Project Schedule, but in many project situations the benefits of conducting a review far outweigh the time considerations. The Project Manager must evaluate the needs of his/her project, determine and document which, if any, deliverables should follow this process, and build the required time and resources into the Project Schedule.

Prior to conducting a peer review, a Project Team member should be identified as the facilitator or person responsible for keeping the review on track. The facilitator should distribute all relevant information pertaining to the deliverable to all participants in advance of the meeting to prepare them to participate effectively.

During the meeting, the facilitator should record information including:

- Peer review date
- Names and roles of participants
- The name of the deliverable being reviewed
- Number of quality issues found

- Description of each quality issue found
- Actions to follow to correct the quality issues prior to presenting the deliverable to the approver
- Names of the individuals responsible for correcting the quality issues
- The date by which quality issues must be corrected

This information should be distributed to the Project Manager, all meeting participants, and those individuals not involved in the meeting who will be responsible for correcting any problems discovered or for producing similar deliverables. The facilitator should also solicit input from the meeting participants to determine if another peer review is necessary. Once the quality issues have been corrected and the Project Manager is confident the deliverable meets expectations, it may be presented to the approver.

Project Deliverables (Project deliverables will differ depending upon the project lifecycle being used. Customize the following questions and add others as necessary to properly and sufficiently evaluate the deliverables specific to your project.)

- Do the deliverables meet the needs of the performing Organization?
- Do the deliverables meet the objectives and goals outlined in the Business Case?
- Do the deliverables achieve the quality standards defined in the Quality Management Plan?

4.8 PROJECT MANAGEMENT DELIVERABLES

- Does the Project Proposal define the business need the project will address, and how the projects product will support the organizations strategic plan?
- Does the Business Case provide an analysis of the costs and benefits of the project and provide a compelling case for the project?
- Has a Project Repository been established to store all project documents, and has it been made available to the Project Team?
- Does the Project Initiation Plan define the project goals and objectives?
- Does the Project Scope provide a list of all the processes that will be affected by the project?

- In the Project Scope, is it clear as to what is in and out of scope?
- Is the Project Schedule defined sufficiently to enable the Project Manager to manage task execution?
- Was a Project Schedule baseline established?
- Is the Project Schedule maintained on a regular basis?
- Does the Quality Management Plan describe quality standards for the project and associated quality assurance and quality control activities?
- Has a project budget been established and documented in sufficient detail?
- Have project risks been identified and prioritized, and has a mitigation plan been developed and documented for each?
- If any risk events have occurred to date, was the risk mitigation plan executed successfully?
- Are all Stakeholders aware of their involvement in the project, and has this it been documented and stored in the project repository?
- Does the Communications Plan describe the frequency and method of communications for all Stakeholders involved in the project?
- Does the Change Control Process describe how to identify change, what individuals may request a change, and the process to follow to approve or reject a request for change?
- Has changes to scope been successfully managed so far?
- Does the Acceptance Management Process clearly define who is responsible for reviewing and approving project and project management deliverables? Does it describe the process to follow to accept or reject deliverables?
- Has the Acceptance Management Process proven successful for the deliverables produced so far?
- Does the Issue Management Process clearly define how issues will be captured, tracked, and prioritized? Does it define the procedure to follow should an unresolved issue need to be escalated?
- Have issues been successfully managed up to this point?
- Does the Organizational Change Management Plan document how changes to people, existing business processes, and culture will be handled?
- Has a Project Team Training Plan been established, and is it being implemented?

- Does the Implementation and Transition Plan describe how to ensure that all Consumers are prepared to use the projects product, and the Performing Organization is prepared to support the product?
- Have all Project Management deliverables been approved by the Project Sponsor and / or Project Director (or designated approver?)
- Does the Project Plan contain all required components as listed in the Guidebook?
- Are each Project Plan component being maintained on a regular basis?

4.9 SUMMARY:

This chapter is based on system feasibility; roadmap and Business system analysis. Feasibility factors make much more effect on the system process and cost.

Questions:

1. Explain common factors for Feasibility study in detail?

   Ans: refer 4.2

2. Discuss the feasibility study factors for Online Examination System in brief.

   Ans.Refer 4.1

❖❖❖❖

5

Modeling system requirements

Unit Structure

5.1 INTRODUCTION:

Requirements analysis in systems engineering and software engineering, encompasses those tasks that go into determining the needs or conditions to meet for a new or altered product, taking account of the possibly conflicting requirements of the various stakeholders, such as beneficiaries or users.

Requirements analysis is critical to the success of a development project.[2] Requirements must be documented, actionable, measurable, testable, related to identified business needs or opportunities, and defined to a level of detail sufficient for system design. Requirements can be functional and non-functional.

Conceptually, requirements analysis includes three types of activity:

- Eliciting requirements: the task of communicating with customers and users to determine what their requirements are. This is sometimes also called requirements gathering.
- Analyzing requirements: determining whether the stated requirements are unclear, incomplete, ambiguous, or contradictory, and then resolving these issues.
- Recording requirements: Requirements might be documented in various forms, such as natural-language documents, use cases, user stories, or process specifications.

Requirements analysis can be a long and arduous process during which many delicate psychological skills are involved. New systems change the environment and relationships between people, so it is important to identify all the stakeholders, take into account all their needs and ensure they understand the implications of the new systems. Analysts can employ several techniques to elicit the requirements from the customer. Historically, this has included such things as holding interviews, or holding focus groups (more aptly named in this context as requirements workshops) and creating requirements lists. More modern techniques include prototyping, and use cases. Where necessary, the analyst will employ a combination of these methods to establish the exact requirements of the stakeholders, so that a system that meets the business needs is produced.

5.2 REQUIREMENT ENGINEERING:

Systematic requirements analysis is also known as requirements engineering.[3] It is sometimes referred to loosely by names such as requirements gathering, requirements capture, or requirements specification. The term requirements analysis can also be applied specifically to the analysis proper, as opposed to elicitation or documentation of the requirements, for instance. Requirements Engineering can be divided into discrete chronological steps:

- Requirements elicitation,
- Requirements analysis and negotiation,
- Requirements specification,
- System modeling
- Requirements validation,
- Requirements management.

Requirement engineering according to Laplante (2007) is "a subdiscipline of systems engineering and software engineering that is concerned with determining the goals, functions, and constraints of hardware and software systems."[4] In some life cycle models, the requirement engineering process begins with a feasibility study activity, which leads to a feasibility report. If the feasibility study suggests that the product should be developed, then requirement analysis can begin. If requirement analysis precedes feasibility studies, which may foster outside the box thinking, then feasibility should be determined before requirements are finalized.

5.3 SOFTWARE REQUIREMENTS SPECIFICATION

A software requirements specification (SRS) is a complete description of the behaviour of the system to be developed. It includes a set of use cases that describe all of the interactions that the users will have with the software. Use cases are also known as functional requirements. In addition to use cases, the SRS also contains non-functional (or supplementary) requirements. Non-functional requirements are requirements which impose constraints on the design or implementation (such as performance requirements, quality standards, or design constraints).

Recommended approaches for the specification of software requirements are described by IEEE 830-1998. This standard describes possible structures, desirable contents, and qualities of a software requirements specification.

Types of Requirements

Requirements are categorized in several ways. The following are common categorizations of requirements that relate to technical management.

Customer Requirements Statements of fact and assumptions that define the expectations of the system in terms of mission objectives, environment, constraints, and measures of effectiveness and suitability (MOE/MOS). The customers are those that perform the eight primary functions of systems engineering, with special emphasis on the operator as the key customer. Operational requirements will define the basic need and, at a minimum, answer the questions posed in the following listing

- Operational distribution or deployment: Where will the system be used?
- Mission profile or scenario: How will the system accomplish its mission objective?
- Performance and related parameters: What are the critical system parameters to accomplish the mission?
- Utilization environments: How are the various system components to be used?
- Effectiveness requirements: How effective or efficient must the system be in performing its mission?
- Operational life cycle: How long will the system be in use by the user?
- Environment: What environments will the system be expected to operate in an effective manner?

5.4 FUNCTIONAL REQUIREMENTS

Functional requirements explain what has to be done by identifying the necessary task, action or activity that must be accomplished. Functional requirements analysis will be used as the top-level functions for functional analysis

Non-functional Requirements

Non-functional requirements are requirements that specify criteria that can be used to judge the operation of a system, rather than specific behaviors.

Performance Requirements

The extent to which a mission or function must be executed; generally measured in terms of quantity, quality, coverage, timeliness or readiness. During requirements analysis, performance (how well does it have to be done) requirements will be interactively developed across all identified functions based on system life cycle factors; and characterized in terms of the degree of certainty in their estimate, the degree of criticality to system success, and their relationship to other requirements.

Design Requirements

The “build to,” “code to,” and “buy to” requirements for products and “how to execute” requirements for processes expressed in technical data packages and technical manuals.

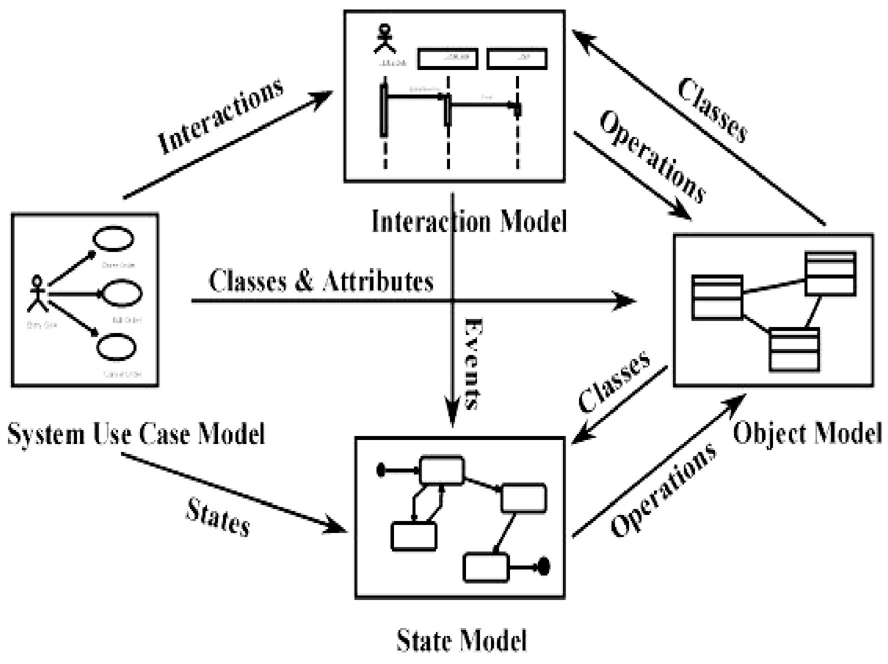
5.5 SYSTEM ANALYSIS MODEL:

The System Analysis Model is made up of class diagrams, sequence or collaboration diagrams and state-chart diagrams. Between them they constitute a logical, implementation-free view of the computer system that includes a detailed definition of every aspect of functionality. This model:

- Defines what the system does not how it does it.
- Defines logical requirements in more detail than the use case model, rather than a physical solution to the requirements.
- Leaves out all technology detail, including system topology

System Model Information Flow:

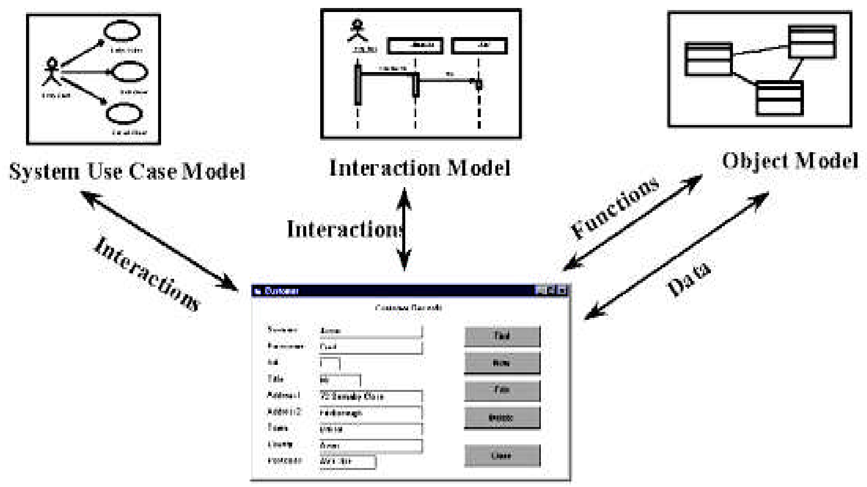
The diagram illustrates the way in which the 3-dimensional system model is developed iteratively from the uses case model in terms of the information which flows between each view. Note that it is not possible fully to develop any one of the three views without the other two. They are interdependent. This is the reason why incremental and iterative development is the most efficient way of developing computer software.



5.6 SCREEN PROTOTYPING:

Screen prototyping can be used as another useful way of getting information from the users. When it is integrated into a UML model:

The flow of the screen is made consistent with the flow of the use case and the interaction model.



The data entered and displayed on the screen is made sistent with the object model.

The functionality of the screen is made consistent with the interaction and object models.

The System Design Model:

This model is the detailed model of everything that is going to be needed to write all the code for the system components. It is the analysis model plus all the implementation detail. Preferably it should be possible automatically to generate at least frame code from this model. This means that any structural changes to the code can be made in the design model and forward generated. This ensures that the design model accurately reflects the code in the components. The design model includes:

(1) Class, sequence or collaboration and state diagrams - as in the analysis model, but now fully defined ready for code generation

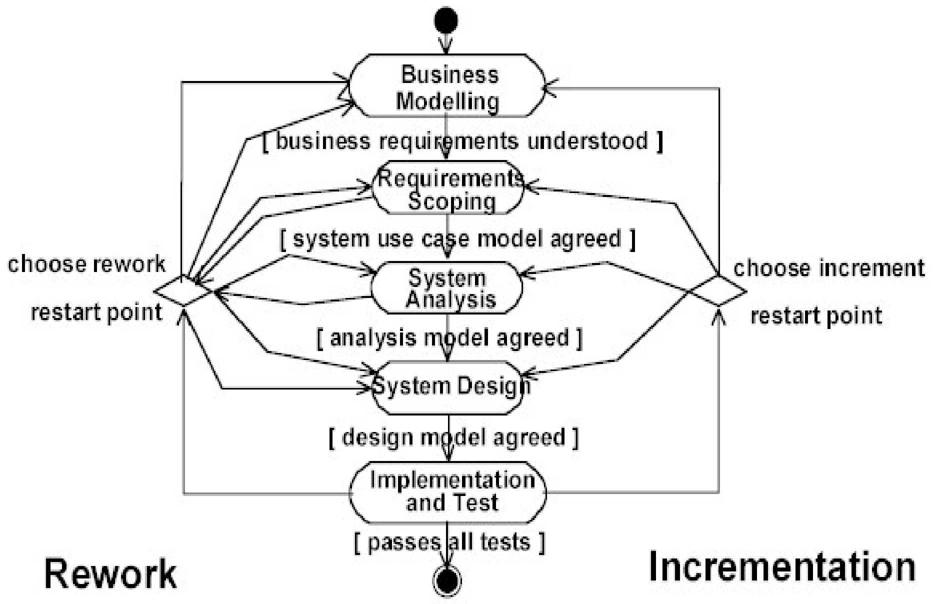
(2) Component diagrams defining all the software components, their interfaces and dependencies

(3) Deployment diagrams defining the topology of the target environment, including which components will run on which computing nodes

Overall Process Flow:

The overall process flow must allow for both rework and incremental development.

Rework - where changes need to be made, the earliest model that the change affects is changed first and the results then flow forward through all the other models to keep them up to date.



Incrementation - increments can restart at any point, depending upon whether the work needed for this increment has already been completed in higher level models.

Incremental Development:

Incremental Development is based on use cases or use case flows which define working pieces of functionality at the user level. Within an 'Increment', the models required to develop a working software increment are each incremented until a working, tested executing piece of software is produced with incremental functionality. This approach:

(1) Improves estimation, planning and assessment. Use cases provide better baselines for estimation than traditionally written specifications. The estimates are continuously updated and improved throughout the project.

(2) Allows risks to the project to be addressed incrementally and reduced early in the lifecycle. Early increments can be scheduled to cover the most risky parts of the architecture. When the architecture is stable, development can be speeded up.

(3) Benefits users, managers and developers who see working functionality early in the lifecycle. Each increment is, effectively, a prototype for the next increment.

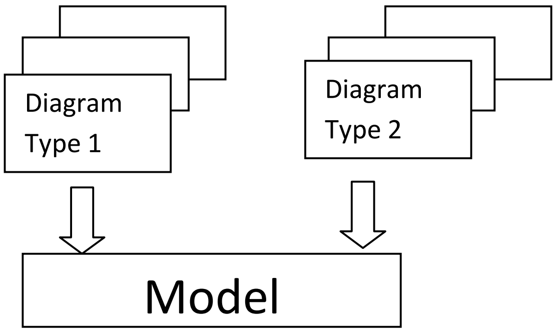
5.7 MODEL ORGANISATION:

All model syntaxes provide a number of model elements which can appear on one or more diagram types. The model elements are contained with a central model, together with all their properties and connections to other model elements. The diagrams are independent views of the model, just as a number of computer screens looking into different records or parts of a database show different views.

The functional view, made up of data flow diagrams, is the primary view of the system. It defines what is done, the flow of data between things that are done and provides the primary structure of the solution. Changes in functionality result in changes in the software structure.

The data view, made up of entity relationship diagrams, is a record of what is in the system, or what is outside the system that is being monitored. It is the static structural view.

The dynamic view, made up of state transition diagrams, defines when things happen and the conditions under which they happen.

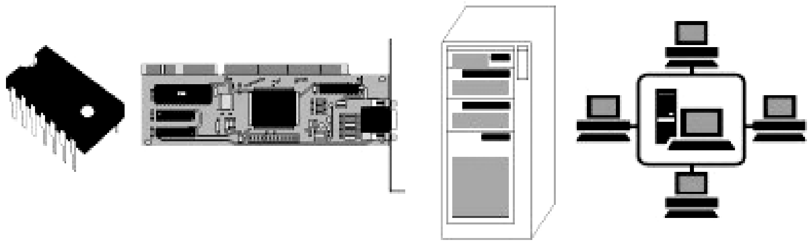


Structured Analysis: In structured analysis there are three orthogonal views:

Encapsulation of Hardware:

The concept of encapsulation of data and functionality that belongs together is something which the hardware industry has been doing for a long time. Hardware engineers have been creating re-useable, re-configurable hardware at each level of abstraction since the early sixties. Elementary Boolean functions are encapsulated together with bits and bytes of data in registers on chips. Chips are encapsulated together on circuit boards. Circuit boards are made to work together in various system boxes that make up the computer. Computers are made to work together across networks. Hardware design, therefore, is totally object oriented at every level and is, as a result, maximally re-useable, extensible and maintainable; in a single word: flexible. Applying object-orientation to software, therefore, could be seen as putting the engineering into software design that has existed in hardware design for many years.

Hardware encapsulates data and function at every level of abstraction

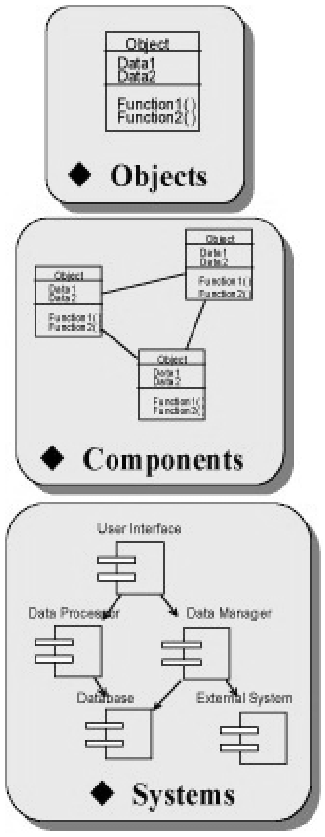


Maximises maintainability, reuse and extension

Encapsulation of Software:

In well developed object oriented software, functionality and data is encapsulated in objects. Objects are encapsulated in components. Components are encapsulated into systems. If this is done well the result is:

Maximal coherence
Minimal interconnection
Solid interface definitions



5.8 SUMMARY

This chapter based on requirement analysis which includes data requirement, data analysis, hardware, software requirements as well as system user requirements.

Questions:

1. Explain Software requirements specification in detail?

Ans: refer 5.3

2. Explain System Analysis Model?

Ans: refer 5.5

❖❖❖❖

6

DATA FLOW DIAGRAMS

Unit Structure

6.1 INTRODUCTION:

A **data-flow diagram** (**DFD**) is a graphical representation of the "flow" of data through an information system. DFDs can also be used for the visualization of data processing (structured design).

On a DFD, data items flow from an external data source or an internal data store to an internal data store or an external data sink, via an internal process.

A DFD provides no information about the timing of processes, or about whether processes will operate in sequence or in parallel. It is therefore quite different from a flowchart, which shows the flow of control through an algorithm, allowing a reader to determine what operations will be performed, in what order, and under what circumstances, but not what kinds of data will be input to and output from the system, nor where the data will come from and go to, nor where the data will be stored (all of which are shown on a DFD).

6.2 OVERVIEW OF DFD:

It is common practice to draw a context-level data flow diagram first, which shows the interaction between the system and external agents which act as data sources and data sinks. On the context diagram (also known as the 'Level 0 DFD') the system's interactions with the outside world are modelled purely in terms of data flows across the *system boundary*. The context diagram shows the entire system as a single process, and gives no clues as to its internal organization.

This context-level DFD is next "exploded", to produce a Level 1 DFD that shows some of the detail of the system being modeled. The Level 1 DFD shows how the system is divided into sub-systems (processes), each of which deals with one or more of the data flows to or from an external agent, and which together provide all of the functionality of the system as a whole. It also identifies internal data stores that must be present in order for the system to do its job, and shows the flow of data between the various parts of the system.

Data-flow diagrams were proposed by Larry Constantine, the original developer of structured design, based on Martin and Estrin's "data-flow graph" model of computation.

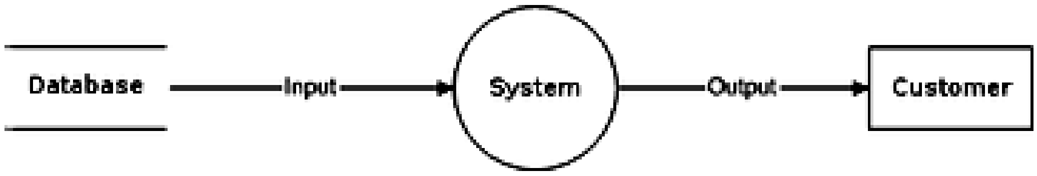
Data-flow diagrams (DFDs) are one of the three essential perspectives of the structured-systems analysis and design method SSADM. The sponsor of a project and the end users will need to be briefed and consulted throughout all stages of a system's evolution. With a data-flow diagram, users are able to visualize how the system will operate, what the system will accomplish, and how the system will be implemented. The old system's dataflow diagrams can be drawn up and compared with the new system's data-flow diagrams to draw comparisons to implement a more efficient system. Data-flow diagrams can be used to provide the end user with a physical idea of where the data they input ultimately has an effect upon the structure of the whole system from order to dispatch to report. How any system is developed can be determined through a data-flow diagram.

6.2.1 How to develop Data Flow Diagram:

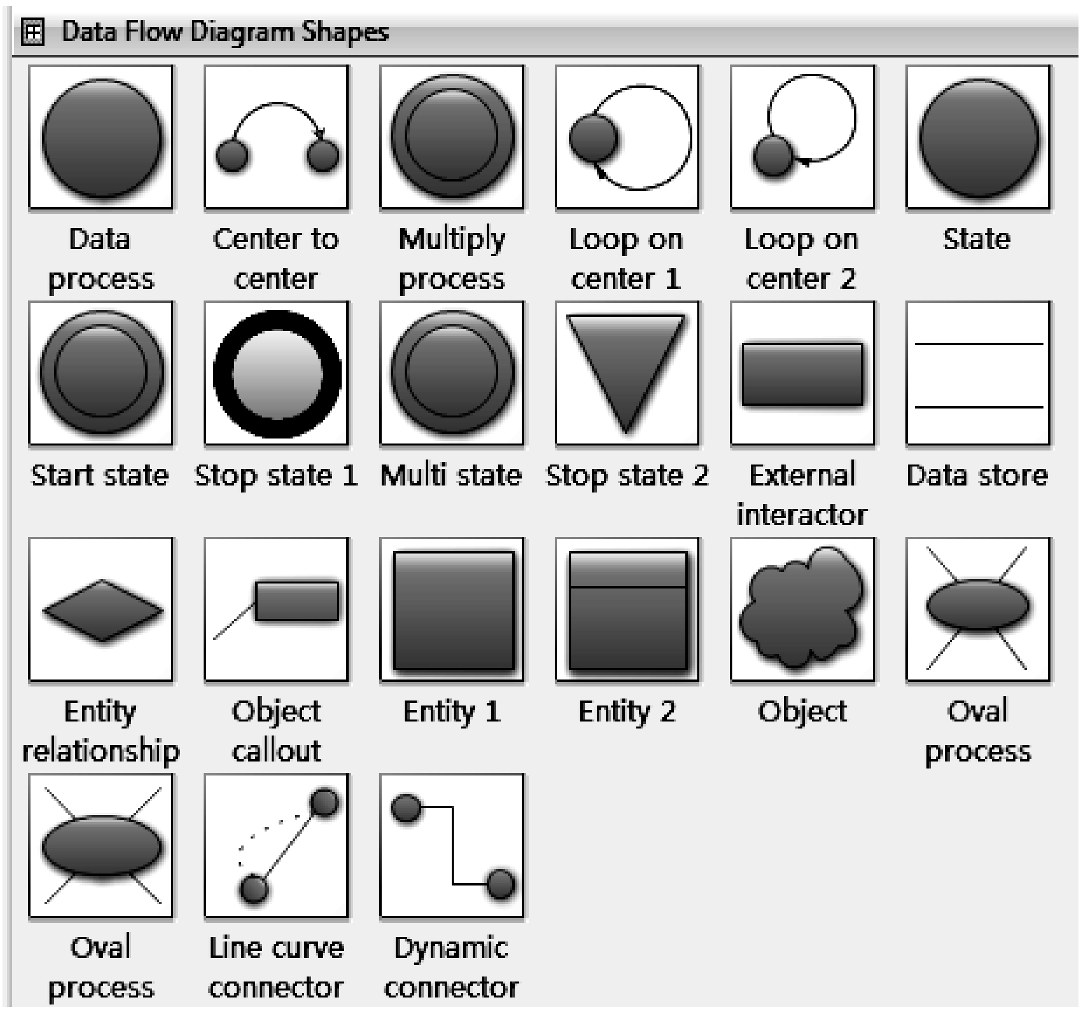
Top-down approach

1. The system designer makes a context level DFD or Level 0, which shows the "interaction" (data flows) between "the system" (represented by one process) and "the system environment" (represented by terminators).
2. The system is "decomposed in lower-level DFD (Level 1)" into a set of "processes, data stores, and the data flows between these processes and data stores".
3. Each process is then decomposed into an "even-lower-level diagram containing its sub processes".
4. This approach "then continues on the subsequent sub processes", until a necessary and sufficient level of detail is reached which is called the primitive process (aka chewable in one bite).

Diagram: Example of DFD flow:



6.2.2 Notation for to Draw Data Flow Diagram:



Data flow diagrams present the logical flow of information through a system in graphical or pictorial form. Data flow diagrams have only four symbols, which makes useful for communication between analysts and users. Data flow diagrams (DFDs) show the data used and provided by processes within a system. DFDs make use of four basic symbols.

Create structured analysis, information flow, process-oriented, data-oriented, and data process diagrams as well as data flowcharts.

External Entity

An external entity is a source or destination of a data flow which is outside the area of study. Only those entities which originate or receive data are represented on a business process diagram. The symbol used is an oval containing a meaningful and unique identifier.

Process

A process shows a transformation or manipulation of data flows within the system. The symbol used is a rectangular box which contains 3 descriptive elements:

Firstly an identification number appears in the upper left hand corner. This is allocated arbitrarily at the top level and serves as a unique reference.

Secondly, a location appears to the right of the identifier and describes where in the system the process takes place. This may, for example, be a department or a piece of hardware. Finally, a descriptive title is placed in the centre of the box. This should be a simple imperative sentence with a specific verb, for example 'maintain customer records' or 'find driver'.

Data Flow

A data flow shows the flow of information from its source to its destination. A data flow is represented by a line, with arrowheads showing the direction of flow. Information always flows to or from a process and may be written, verbal or electronic. Each data flow may be referenced by the processes or data stores at its head and tail, or by a description of its contents.

Data Store

A data store is a holding place for information within the system:

It is represented by an open ended narrow rectangle. Data stores may be long-term files such as sales ledgers, or may be short-term accumulations: for example batches of documents that are waiting to be processed. Each data store should be given a reference followed by an arbitrary number.

Resource Flow

A resource flow shows the flow of any physical material from its source to its destination. For this reason they are sometimes referred to as physical flows.

The physical material in question should be given a meaningful name. Resource flows are usually restricted to early, high-level diagrams and are used when a description of the physical flow of materials is considered to be important to help the analysis.

External Entities

It is normal for all the information represented within a system to have been obtained from, and/or to be passed onto, an external source or recipient. These external entities may be duplicated on a diagram, to avoid crossing data flow lines. Where they are duplicated a stripe is drawn across the left hand corner, like this.

The addition of a lowercase letter to each entity on the diagram is a good way to uniquely identify them.

Processes

When naming processes, avoid glossing over them, without really understanding their role. Indications that this has been done are the use of vague terms in the descriptive title area - like 'process' or 'update'.

The most important thing to remember is that the description must be meaningful to whoever will be using the diagram.

Data Flows

Double headed arrows can be used (to show two-way flows) on all but bottom level diagrams. Furthermore, in common with most of the other symbols used, a data flow at a particular level of a diagram may be decomposed to multiple data flows at lower levels.

Data Stores

Each store should be given a reference letter, followed by an arbitrary number. These reference letters are allocated as follows:
'D' - indicates a permanent computer file
'M' - indicates a manual file
'T' - indicates a transient store, one that is deleted after processing.
In order to avoid complex flows, the same data store may be drawn several times on a diagram. Multiple instances of the same data store are indicated by a double vertical bar on their left hand edge.

6.2.3 Data Flow Diagram Example:

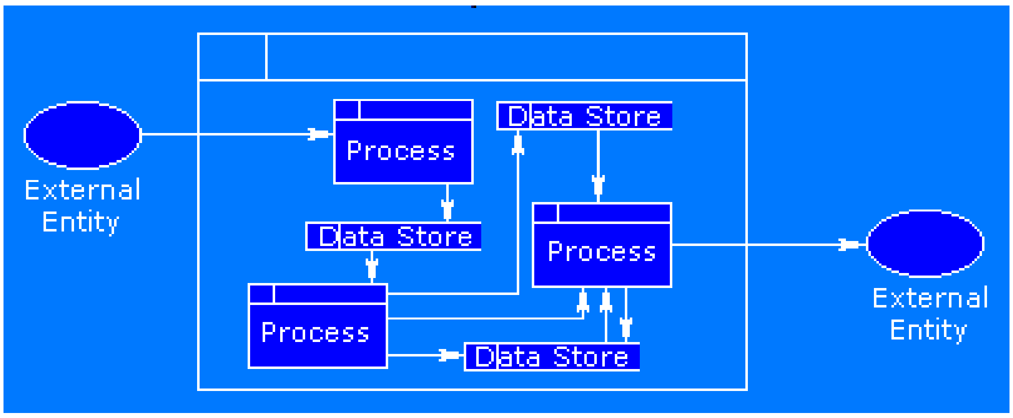
A **data flow Diagram example** is a graphic representation of all the major steps of a process. It can help you:

• Understand the complete process.
• Identify the critical stages of a process.
• Locate problem areas.
• Show relationships between different steps in a process.

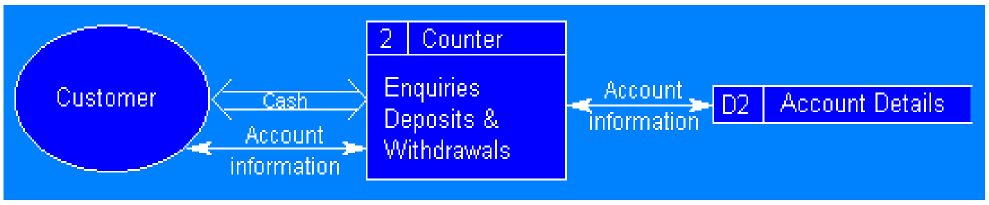
Professional looking examples and templates of Data Flow Diagram which help you create data flow diagrams rapidly. Document 6 Sigma and ISO 9000 processes.

Example:

Data flow diagrams can be used to provide a clear representation of any business function. The technique starts with an overall picture of the business and continues by analyzing each of the functional areas of interest. This analysis can be carried out to precisely the level of detail required. The technique exploits a method called top-down expansion to conduct the analysis in a targeted way.

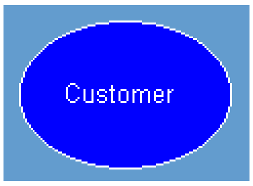


There are only five symbols that are used in the drawing of business process diagrams (data flow diagrams). These are now explained, together with the rules that apply to them.



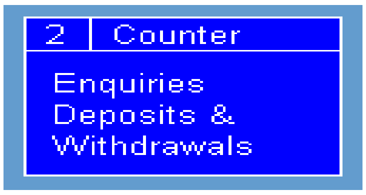
This diagram represents a banking process, which maintains customer accounts. In this example, customers can withdraw or deposit cash, request information about their account or update their account details. The five different symbols used in this example represent the full set of symbols required to draw any business process diagram.

External Entity



An external entity is a source or destination of a data flow which is outside the area of study. Only those entities which originate or receive data are represented on a business process diagram. The symbol used is an oval containing a meaningful and unique identifier.

Process

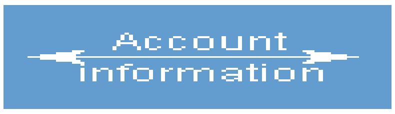


A process shows a transformation or manipulation of data flows within the system. The symbol used is a rectangular box which contains 3 descriptive elements:

Firstly an identification number appears in the upper left hand corner. This is allocated arbitrarily at the top level and serves as a unique reference.

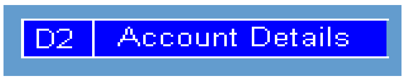
Secondly, a location appears to the right of the identifier and describes where in the system the process takes place. This may, for example, be a department or a piece of hardware. Finally, a descriptive title is placed in the centre of the box. This should be a simple imperative sentence with a specific verb, for example 'maintain customer records' or 'find driver'.

Data Flow



A data flow shows the flow of information from its source to its destination. A data flow is represented by a line, with arrowheads showing the direction of flow. Information always flows to or from a process and may be written, verbal or electronic. Each data flow may be referenced by the processes or data stores at its head and tail, or by a description of its contents.

Data Store



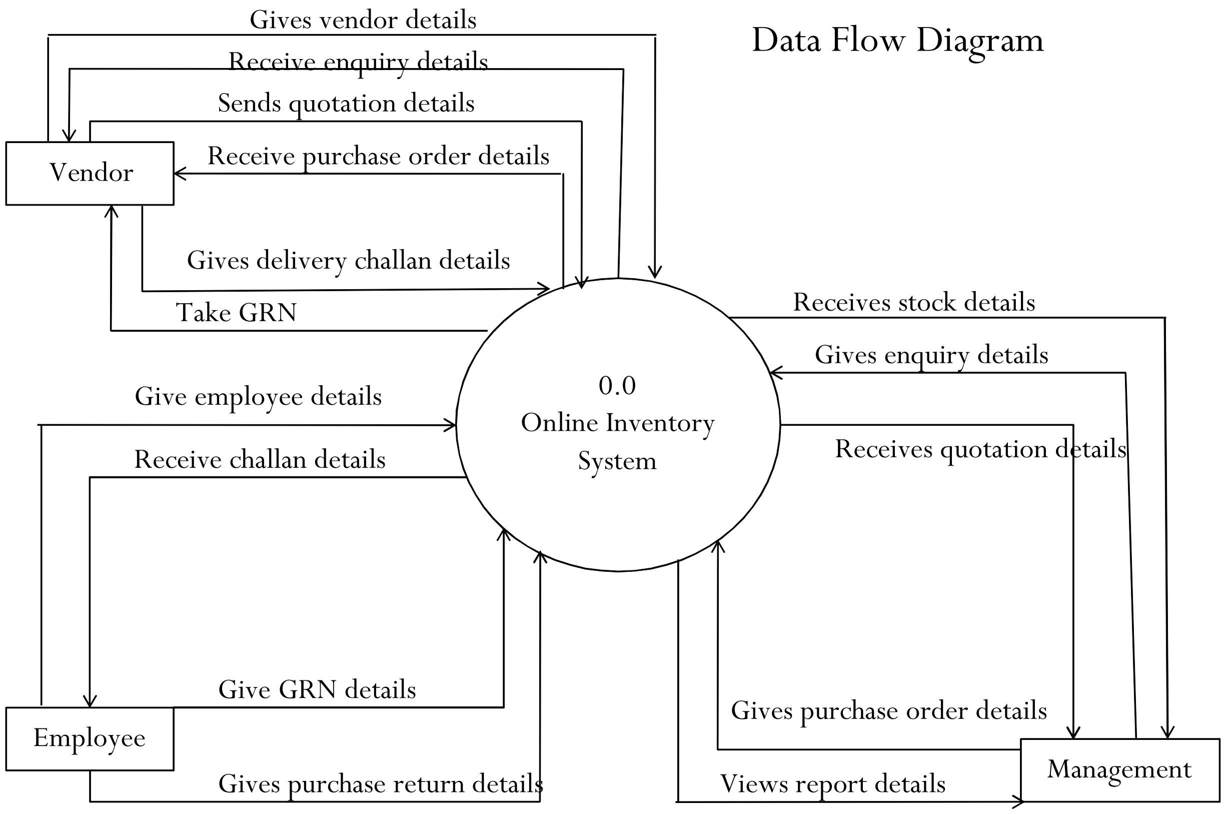
A data store is a holding place for information within the system: It is represented by an open ended narrow rectangle. Data stores may be long-term files such as sales ledgers, or may be short-term accumulations: for example batches of documents that are waiting to be processed. Each data store should be given a reference followed by an arbitrary number.

ResourceFlow

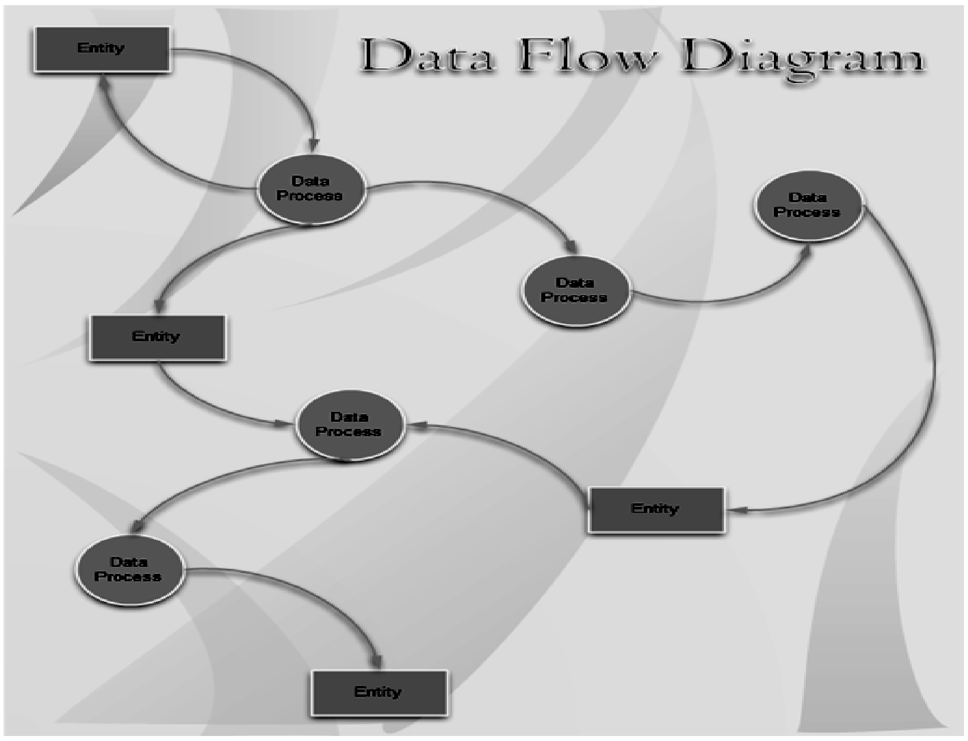
A resource flow shows the flow of any physical material from its source to its destination. For this reason they are sometimes referred to as physical flows.

The physical material in question should be given a meaningful name. Resource flows are usually restricted to early, high-level diagrams and are used when a description of the physical flow of materials is considered to be important to help the analysis.

Example for Online Inventory System:



Examples of Data Flow Diagram



6.3 SUMMARY:

This chapter based on systematic flow of data with the help of some notation, symbol which established by scientist. The flow of data indicates sequence to execute and process data in the system. This DFD is an analysis method to analyze the data up to user satisfied level.

Questions:

1.Explain DFD with example?

Ans: refer 6.2

2.Draw a DFD for Sale & Purchase Management System for Manufacturing Industry.

Ans: refer 6.2

3. .Draw a DFD for Event Management System for Hotel.

Ans: refer 6.2

❖❖❖❖

7

ENTITY RELATIONSHIP DIAGRAM

Unit Structure

7.1 INTRODUCTION:

This activity is designed to help you understand the process of designing and constructing ERDs using Systems Architect. Entity Relationship Diagrams are a major data modeling tool and will help organize the data in your project into entities and define the relationships between the entities. This process has proved to enable the analyst to produce a good database structure so that the data can be stored and retrieved in a most efficient manner.

7.2 ENTITY

A data entity is anything real or abstract about which we want to store data. Entity types fall into five classes: roles, events, locations, tangible things or concepts. E.g. employee, payment, campus, book. Specific examples of an entity are called **instances.** E.g. the employee John Jones, Mary Smith's payment, etc.

Relationship

A data relationship is a natural association that exists between one or more entities. E.g. Employees process payments. **Cardinality** defines the number of occurrences of one entity for a single occurrence of the related entity. E.g. an employee may process many payments but might not process any payments depending on the nature of her job.

Attribute

A data attribute is a characteristic common to all or most instances of a particular entity. Synonyms include property, data element, field. E.g. Name, address, SSN, pay rate are all attributes of the entity employee. An attribute or combination of attributes that uniquely identifies one and only one instance of an entity is called a **primary key** or **identifier**. E.g. SSN is a primary key for Employee.

7.3 ENTITY RELATIONSHIP DIAGRAM METHODOLOGY

1. Identify Entities	Identify the roles, events, locations, tangible things or concepts about which the end-users want to store data.
2. Find Relationships	Find the natural associations between pairs of entities using a relationship matrix.
3. Draw Rough ERD	Put entities in rectangles and relationships on line segments connecting the entities.
4. Fill in Cardinality	Determine the number of occurrences of one entity for a single occurrence of the related entity.
5. Define Primary Keys	Identify the data attribute(s) that uniquely identify one and only one occurrence of each entity.
6. Draw Key-Based ERD	Eliminate Many-to-Many relationships and include primary and foreign keys in each entity.
7. Identify Attributes	Name the information details (fields) which are essential to the system under development.

8. Map Attributes	For each attribute, match it with exactly one entity that it describes.
9. Draw fully attributed ERD	Adjust the ERD from step 6 to account for entities or relationships discovered in step 8.
10. Check Results	Does the final Entity Relationship Diagram accurately depict the system data?

7.3.1 Solved Example:

A company has several departments. Each department has a supervisor and at least one employee. Employees must be assigned to at least one, but possibly more departments. At least one employee is assigned to a project, but an employee may be on vacation and not assigned to any projects. The important data fields are the names of the departments, projects, supervisors and employees, as well as the supervisor and employee SSN and a unique project number.

1. Identify Entities

The entities in this system are Department, Employee, Supervisor and Project. One is tempted to make Company an entity, but it is a false entity because it has only one instance in this problem. True entities must have more than one instance.

2. Find Relationships:

We construct the following Entity Relationship Matrix:

Dept.	Employee	Supervisor	Project
Department		is assigned	run by
Employee		belongs to	Works on
Supervisor	runs	Project	Uses

3. Draw Rough ERD:

We connect the entities whenever a relationship is shown in the entity Relationship Matrix.

```
__________                                                ________
| Department |______________Run by___  _____________| Supervisor |
|            |                                        |            |
  --------------                                      --------------
  |
  |
  |           ------------                          -----------
  |___Is_____| Employee |_______Works on__________| Project |
   assigned   |            |                          |           |
              ------------                          -----------
```

4. Fill in Cardinality

From the description of the problem we see that:

- Each department has exactly one supervisor.
- A supervisor is in charge of one and only one department.
- Each department is assigned at least one employee.
- Each employee works for at least one department.
- Each project has at least one employee working on it.
- An employee is assigned to 0 or more projects.

```
____________                                        ___________
| Department |              Run by                  | Supervisor |
|            |-++-----------------------++-|   |
--|-----------                                      --------------
  W
  +
  |          ------------                       -----------
  |  Is   | Employee |    Works on              | Project |
  --------+<-|               |->+----------------------0<-|      |
   Assigned  ------------                        -----------
```

5. Define Primary Keys

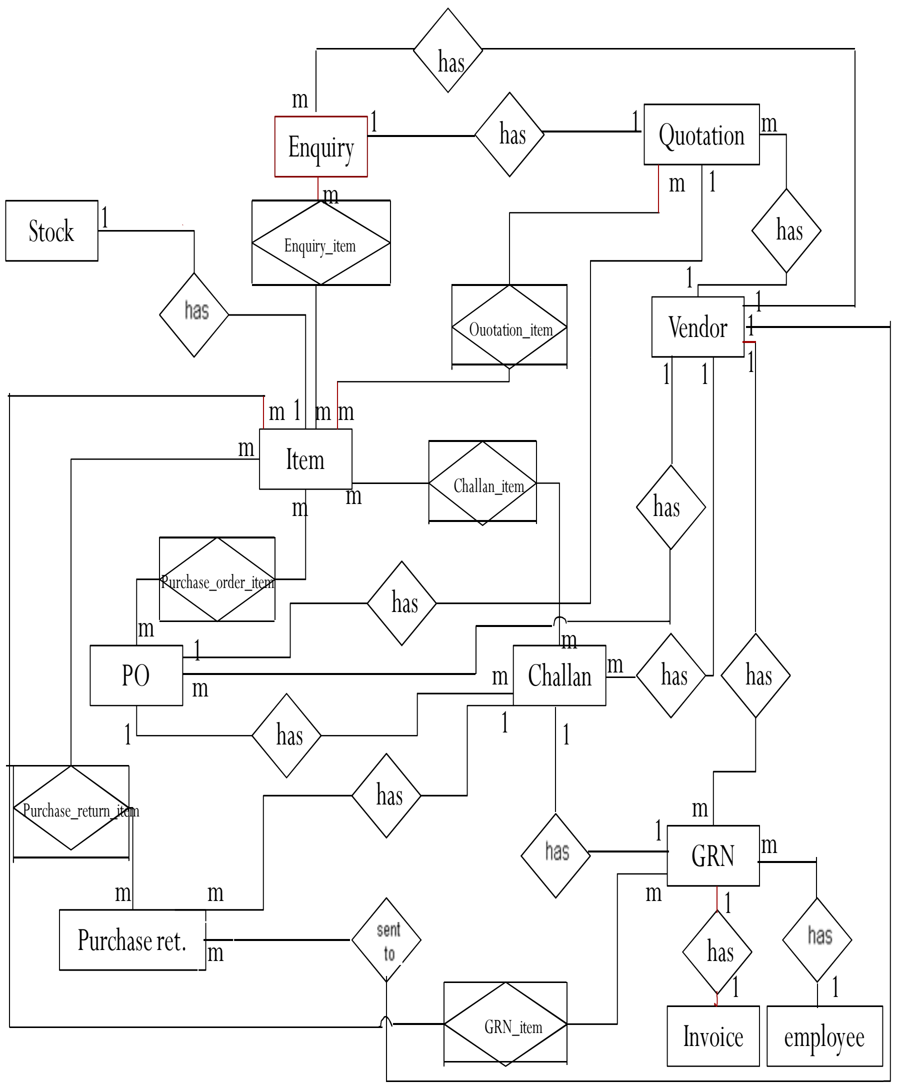
The primary keys are Department Name, Supervisor SSN, Employee SSN, Project Number.

6. Draw Key-Based ERD

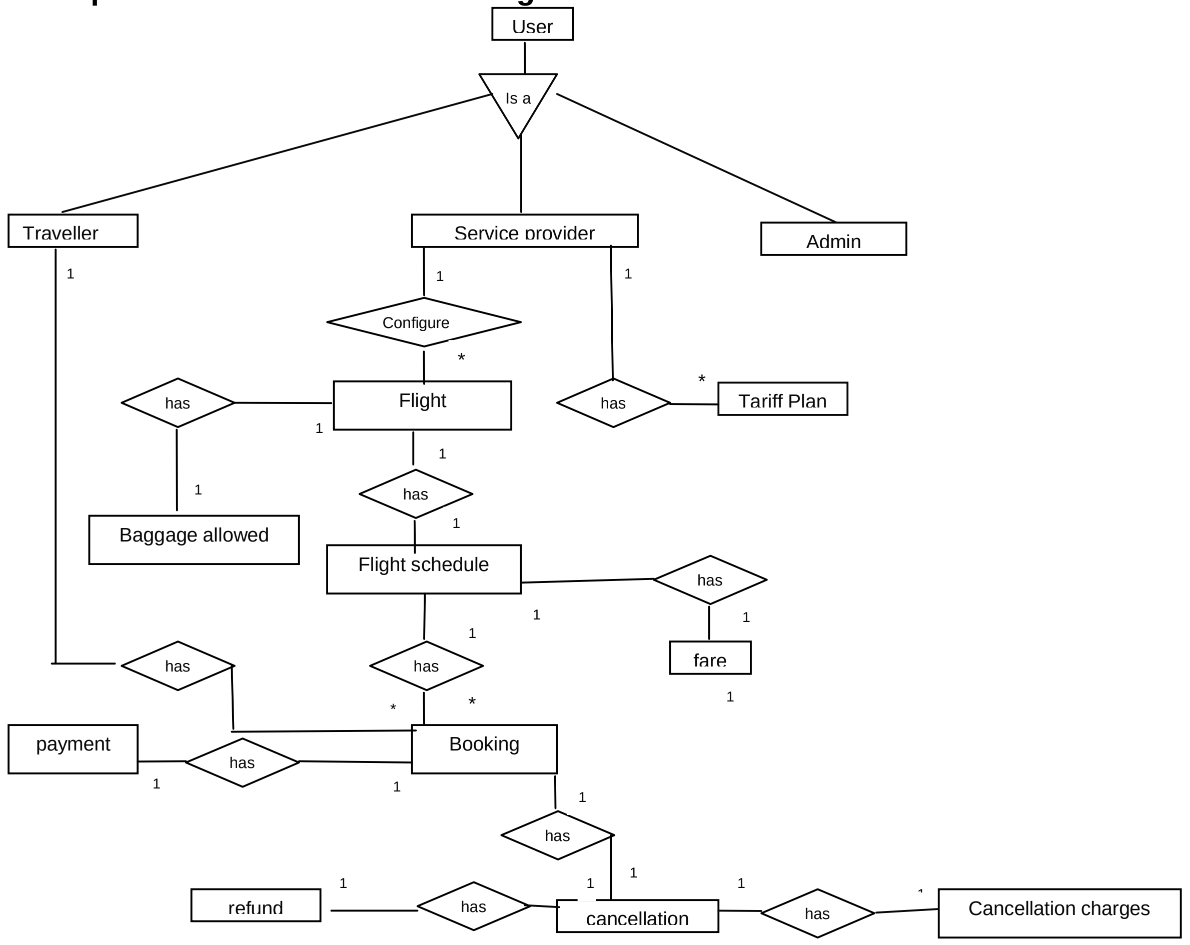
There are two many-to-many relationships in the rough ERD above, between Department and Employee and between Employee and Project. Thus we need the associative entities Department-Employee and Employee-Project. The primary key for Department-Employee is the concatenated key Department Name and Employee SSN. The primary key for Employee-Project is the concatenated key Employee SSN and Project Number.

Example for Inventory Management System:

ERD:



Example to draw ERD for Travelling Service:



7. Identify Attributes

The only attributes indicated are the names of the departments, projects, supervisors and employees, as well as the supervisor and employee SSN and a unique project number.

8. Map Attributes

Attribute	Entity	Attribute	Entity
Department Name	Department	Supervisor SSN	Supervisor
Employee SSN	Employee	Supervisor Name	Supervisor
Employee Name	Employee	Project Name	Project
		Project ID	Project

9. Draw Fully Attributed ERD

```
__________________                          9. Draw Fully Attributed
ERD
_______________                              ___________________
| Department   |              Run by          | Supervisor       |
|___Key Data___|-++-------------------------++-|___Key Data________|
|Dept Name[PK1]|                              |Supervisor SSN[PK1]|
--|------------                               |____Non-Key Data___|
 +                                                | Supervisor Name   |
 +                                                  ---------------------
 |
 |         --------------------              -------------------
 |   Is    | Employee-Dept    |   involves | Employee    |
 --------+<-|____Key Data______|->+--------------++-|____Key Data___|
  Assigned  |Dept Name[PK1][FK]|                  |Employee SSN[PK1]|
            |Emp. SSN[PK1][FK] |                 |___Non-Key DATA__|
            --------------------          | Employee Name   |
                                                     -------------------
                                                              +
                                                              +
                                                         Works|on
                                                              |
                                                              0
                                                              M
   --------------------                      -------------------
   |    Project       |     Includes          |Employee-Project |
   |____Key Data______|++-------------------+<-|____Key Data_____|
   |Project Num.[PK1] |                       |Emp. SSN[PK1][FK]|
   |___Non-Key Data___|                       |Proj Num[PK1][FK]|
   | Project Name     |                       -------------------
   --------------------
```

10. Check Results

The final ERD seems to model the data in this system well.

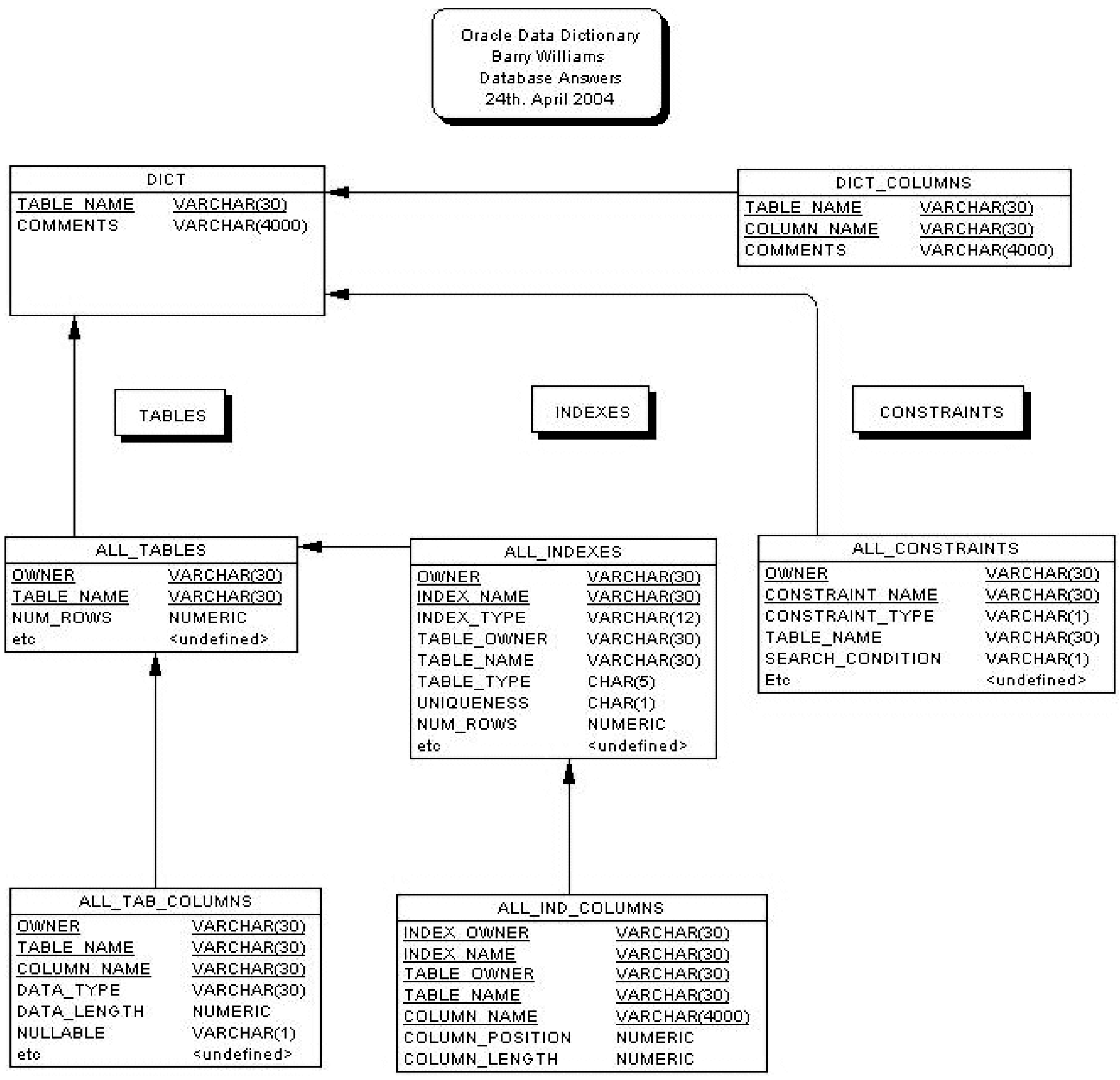
7.4 DATA DICTIONARY:

A **data dictionary**, a.k.a. metadata repository, as defined in the IBM Dictionary of Computing, is a "centralized repository of information about data such as meaning, relationships to other data, origin, usage, and format." The term may have one of several closely related meanings pertaining to databases and database management systems (DBMS):

- a document describing a database or collection of databases
- an integral component of a DBMS that is required to determine its structure
- a piece of middleware that extends or supplants the native data dictionary of a DBMS

Database users and application developers can benefit from an authoritative data dictionary document that catalogs the organization, contents, and conventions of one or more databases. This typically includes the names and descriptions of various tables and fields in each database, plus additional details, like the type and length of each data element. There is no universal standard as to the level of detail in such a document, but it is primarily a weak kind of data.

7.4.1 Example of Data Dictionary:



7.5 UML DIAGRAM:

Unified Modelling Language (**UML**) is a standardized general-purpose modelling language in the field of software engineering. The standard is managed, and was created by, the Object Management Group.

The Unified Modelling Language (UML) is used to specify, visualize, modify, construct and document the artifacts of an object-oriented software intensive system under development.[1] UML offers a standard way to visualize a system's architectural blueprints, including elements such as:

- actors
- business processes
- (logical) components
- activities
- programming language statements
- database schemas, and
- reusable software components

7.5.1 What is UML?

The Unified Modelling Language was originally developed at Rational Software but is now administered by the Object Management Group (see link). It is a modelling syntax aimed primarily at creating models of software-based systems, but can be used in a number of areas. It is:

Syntax only - UML is just a language; it tells you what model elements and diagrams are available and the rules associated with them. It does not tell you what diagrams to create.

Process-independent - the process by which the models are created is separate from the definition of the language. You will need a process in addition to the use of UML itself.

Tool-independent - UML leaves plenty of space for tool vendors to be creative and add value to visual modelling with UML. However, some tools will be better than others for particular applications.

Well documented - the UML notation guide is available as a reference to all the syntax available in the language.

Its application is not well understood - the UML notation guide is not sufficient to teach you how to use the language. It is a generic modelling language and needs to be adapted by the user to particular applications.

Originally just for system modelling - some user-defined extensions are becoming more widely used now, for example, for business modelling and modelling the design of web-based applications.

7.5.2 How UML Started?

UML came about when James Rumbaugh joined Grady Booch at Rational Software. They both had object-oriented syntaxes and needed to combine them. Semantically, they were very similar; it was mainly the symbols that needed to be unified.

UML Diagram is divided into four types:

-Activity Diagrams - a generic flow chart used much in business modelling and sometimes in use case modelling to indicate the overall flow of the use case. This diagram type replaces the need for dataflow diagrams but is not a main diagram type for the purposes of analysis and design.

-State Machine Diagrams - in information systems these tend to be used to descibe the lifecycle of an important data entity. In real-time systems they tend to be used to describe state dependent behaviour

- Component Diagrams - show the types of components, their interfaces and dependencies in the software architecture that is the solution to the application being developed.

- Deployment Diagrams - show actual computing nodes, their communication relationships and the processes or components that run on them.

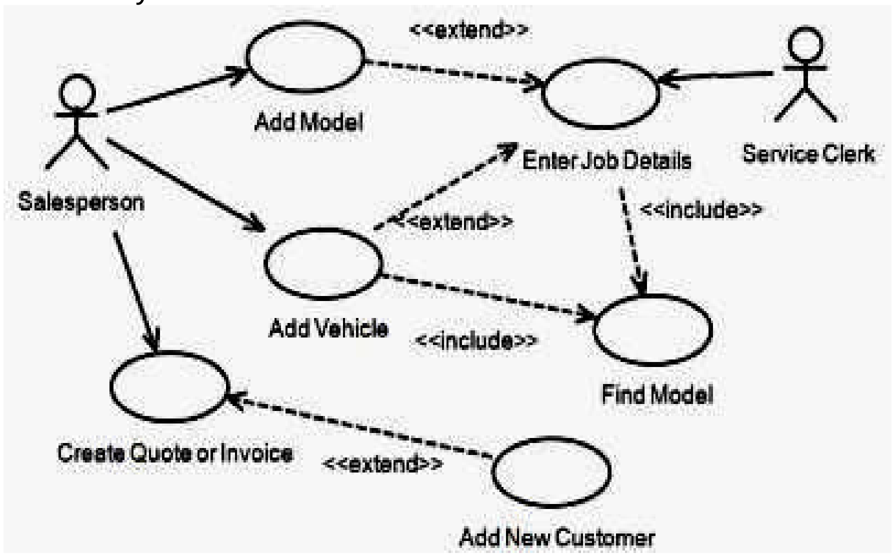
UML can be used to model a business, prior to automating it with computers. The same basic UML syntax is used; however, a number of new symbols are added, in order to make the diagrams more relevant to the business process world. A commonly-used set of these symbols is available in current versions of Rational Rose.

The most commonly used UML extensions for web applications were developed by Jim Conallen. You can access his own website to learn more about them by following the link. These symbols are also available in current versions of Rational Rose.

UML is designed to be extended in this way. Extensions to the syntax are created by adding 'stereotypes' to a model element. The stereotype creates a new model element from an existing one with an extended, user-defined meaning. User defined symbols, which replace the original UML symbol for the model element, can then be assigned to the stereotype. UML itself uses this mechanism, so it is important to know what stereotypes are predefined in UML in order not to clash with them when creating new ones.

The use case diagram shows the functionality of the system from an outside-in viewpoint.

-Actors (stick men) are anything outside the system that interacts with the system.



- Use Cases (ovals) are the procedures by which the actors interact with the system.

- Solid lines indicate which actors interact with the system as part of which procedures.

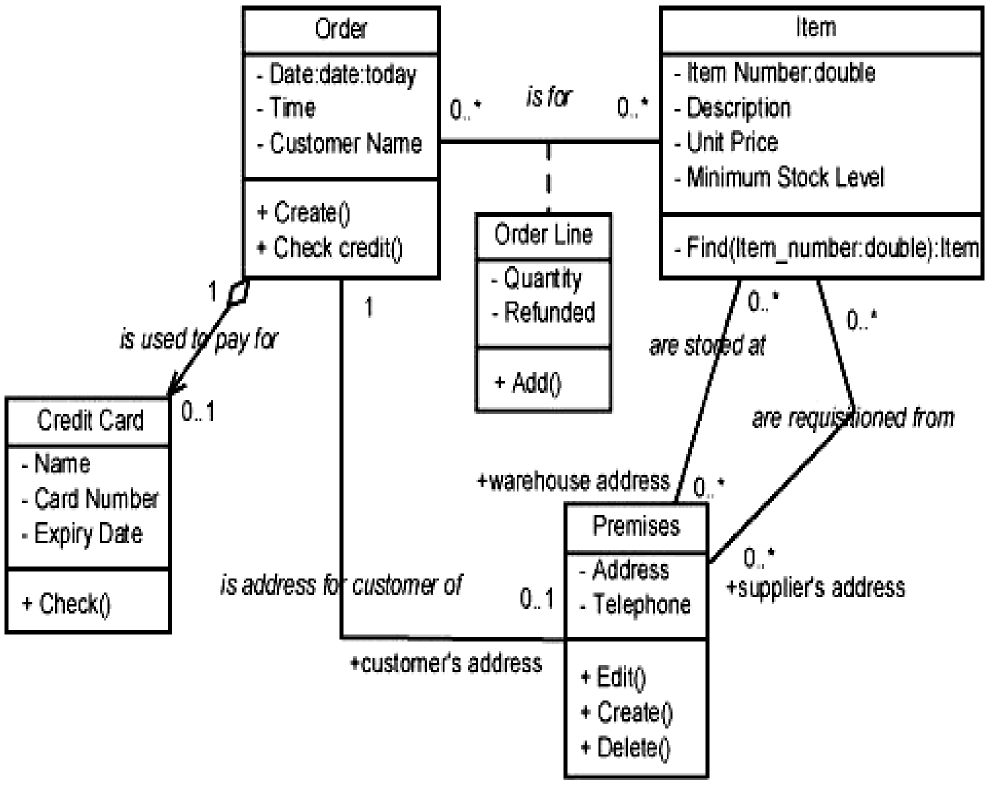
- Dashed lines show dependencies between use cases, where one use case is 'included' in or 'extends' another.

7.6 CLASS DIAGRAM:

Class diagrams show the static structure of the systems. Classes define the properties of the objects which belong to them. These include:

-Attributes - (second container) the data properties of the classes including type, default value and constraints.

7.6.1 Example:



- Operations - (third container) the signature of the functionality that can be applied to the objects of the classes including parameters, parameter types, parameter constraints, return types and the semantics.

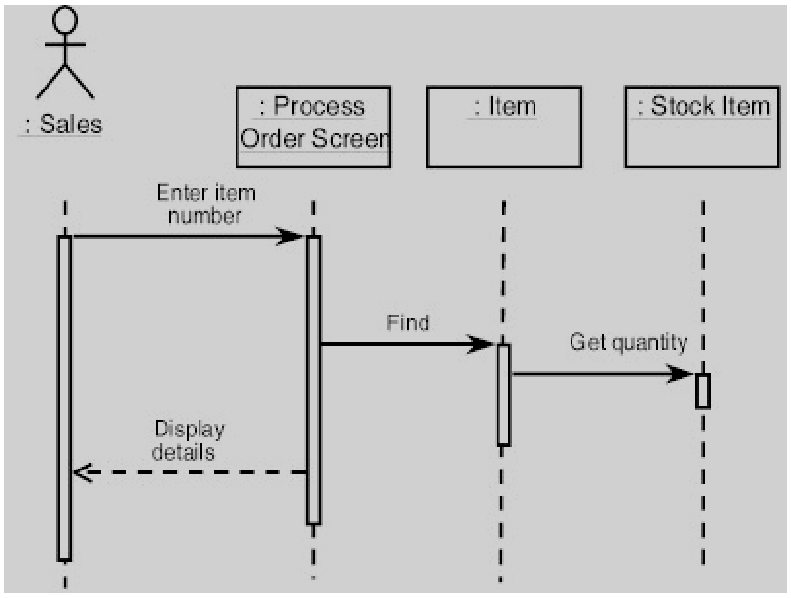
- Associations - (solid lines between classes) the references, contained within the objects of the classes, to other objects, enabling interaction with those objects.

Sequence Diagram:

Sequence diagrams show potential interactions between objects in the system being defined. Normally these are specified as part of a use case or use case flow and show how the use case will be implemented in the system. They include:

- Objects - oblong boxes or actors at the top - either named or just shown as belonging to a class, from, or to which messages are sent to other objects.

Example:



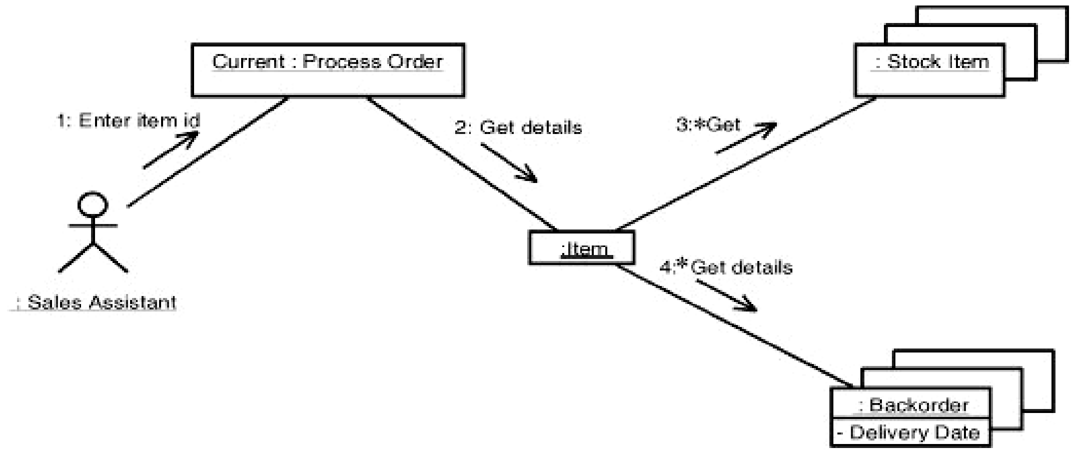
- Messages - solid lines for calls and dotted lines for data returns, showing the messages that are sent between objects including the order of the messages which is from the top to the bottom of the diagram.

- Object lifelines - dotted vertical lines showing the lifetime of the objects.

- Activation - the vertical oblong boxes on the object lifelines showing the thread of control in a synchronous system.

7.7 COMMUNICATION DIAGRAM:

Communication Diagrams show similar information to sequence diagrams, except that the vertical sequence is missing. In its place are:



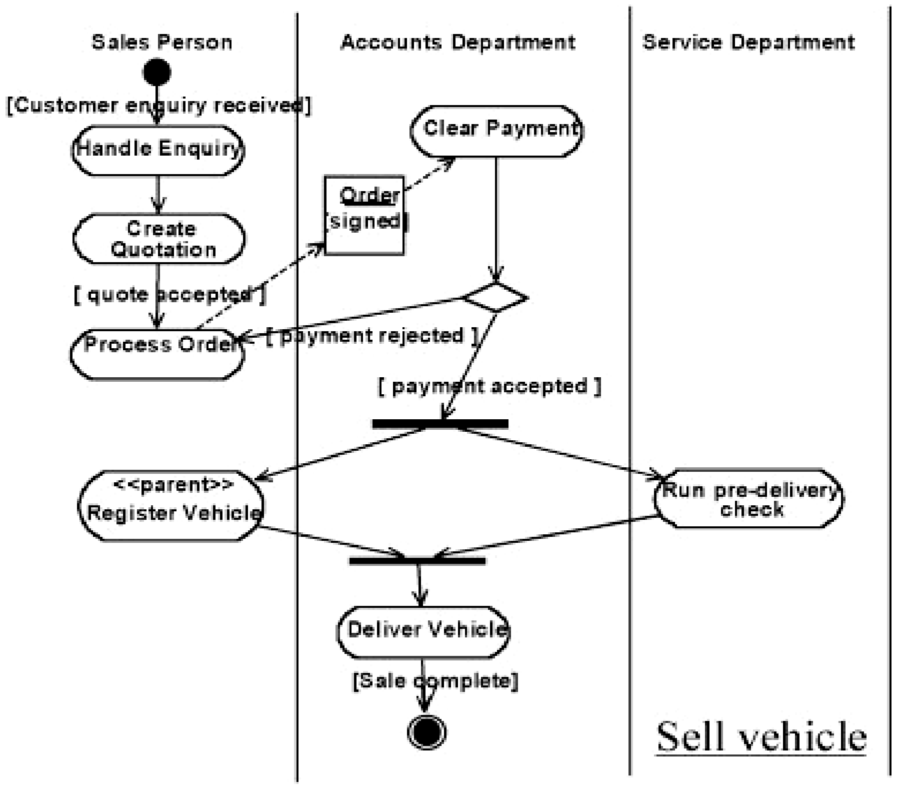
- Object Links - solid lines between the objects. These represent the references between objects that are needed for them to interact and so show the static structure at object level.

- Messages - arrows with one or more message name that show the direction and names of the messages sent between objects.

7.8 ACTIVITY DIAGRAM:

A UML Activity Diagram is a general purpose flowchart with a few extras. It can be used to detail a business process, or to help define complex iteration and selection in a use case description. It includes:

- Active states - oblongs with rounded corners which describe what is done.



- Transitions - which show the order in which the active states occur and represent a thread of activity?

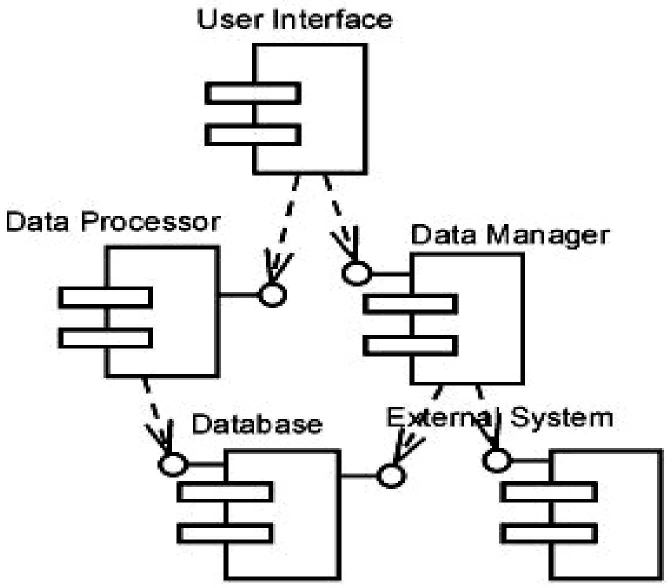
- Conditions - (in square brackets) which qualify the transitions.

-Decisions - (nodes in the transitions) which cause the thread to select one of multiple paths.

7.9 COMPONENT DIAGRAM:

Component Diagrams show the types of software components in the system, their interfaces and dependencies.

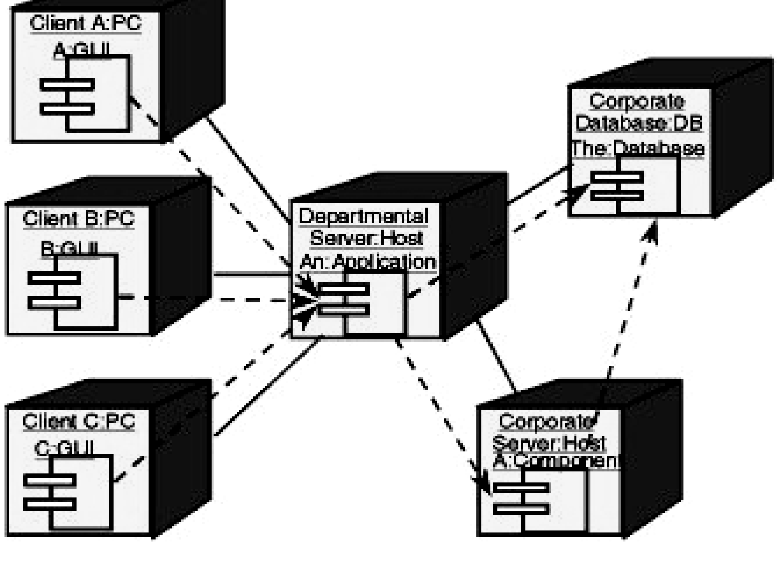
7.9.1 Example:



7.10 DEPLOYMENT DIAGRAM:

Deployment diagrams show the computing nodes in the system, their communication links, the components that run on them and their dependencies.

7.10.1 Example:



7.11 SUMMARY :

In this chapter we discussed many points related to system development and analysis model. These model helps to us to represent any system like online and offline system.

Questions:

1.Explain ER- Diagram in detail?

Ans: refer 7.2

2.Explain UML diagram in detail?

Ans: refer 7.5

3.Draw an ERD for Hotel Mangament System.

Ans: refer 7.2

4.Draw a Use case diagram for Online Shopping for Greeting card.

Ans: refer 7.5

❖❖❖❖

8

DESIGN

Unit Structure

8.1 INTRODUCTION:

Systems design is the process or art of defining the architecture, components, modules, interfaces, and data for a system to satisfy specified requirements. One could see it as the application of systems theory to product development. There is some overlap with the disciplines of systems analysis, systems architecture and systems engineering.

Object-oriented analysis and design methods are becoming the most widely used methods for computer system design. The UML has become the standard language used in Object-oriented analysis and design. It is widely used for modeling software systems and is increasingly used for high designing non-software systems and organizations.

8.2 LOCAL DESIGN:

The logical design of a system pertains to an abstract representation of the data flows, inputs and outputs of the system. This is often conducted via modelling, which involves a simplistic (and sometimes graphical) representation of an actual system. In the context of systems design, modelling can undertake the following forms, including:

- Data flow diagrams
- Entity Life Histories
- Entity Relationship Diagrams

8.3 PHYSICAL DESIGN:

The physical design relates to the actual input and output processes of the system. This is laid down in terms of how data is inputted into a system, how it is verified/authenticated, how it is processed, and how it is displayed as output.

Physical design, in this context, does not refer to the tangible physical design of an information system. To use an analogy, a personal computer's physical design involves input via a keyboard, processing within the CPU, and output via a monitor, printer, etc. It would not concern the actual layout of the tangible hardware, which for a PC would be a monitor, CPU, motherboard, hard drive, modems, video/graphics cards, USB slots, etc

8.4 ALTERNATIVE DESIGN METHOD:

8.4.1 Rapid Application Development (RAD)

Rapid Application Development (RAD) is a methodology in which a systems designer produces prototypes for an end-user. The end-user reviews the prototype, and offers feedback on its suitability. This process is repeated until the end-user is satisfied with the final system.

8.4.2 Joint Application Development (JAD)

JAD is a methodology which evolved from RAD, in which a systems designer consults with a group consisting of the following parties:

- Executive sponsor
- Systems Designer
- Managers of the system

8.5 EMBEDDED SYSTEM:

An **embedded system** is a computer system designed to perform one or a few dedicated functions often with real-time computing constraints. It is embedded as part of a complete device often including hardware and mechanical parts. By contrast, a general-purpose computer, such as a personal computer (PC), is designed to be flexible and to meet a wide range of end-user needs.

Characteristics:

1. Embedded systems are designed to do some specific task, rather than be a general-purpose computer for multiple tasks. Some also have real-time performance constraints that must be met, for reasons such as safety and usability; others may have low or no performance requirements, allowing the system hardware to be simplified to reduce costs.

2. Embedded systems are not always standalone devices. Many embedded systems consist of small, computerized parts within a larger device that serves a more general purpose. For example, the Gibson Robo Guitar features an embedded system for tuning the strings, but the overall purpose of the Robot Guitar is, of course, to play music. Similarly, an embedded system in an automobile provides a specific function as a subsystem of the car itself.

3. The program instructions written for embedded systems are referred to as firmware, and are stored in read-only memory or Flash memory chips. They run with limited computer hardware resources: little memory, small or non-existent keyboard and/or screen.

8.6 DESIGN PHASE ACTIVITIES:

1.The Systems Development Life Cycle (SDLC), or Software Development Life Cycle in systems engineering, information systems and software engineering, is the process of creating or altering systems, and the models and methodologies that people use to develop these systems. The concept generally refers to computer or information systems.

2. Work breakdown structure organization:

The upper section of the Work Breakdown Structure (WBS) should identify the major phases and milestones of the project in a summary fashion. In addition, the upper section should provide an overview of the full scope and timeline of the project and will be part of the initial project description effort leading to project approval.

8.7 BASELINES IN THE SDLC:

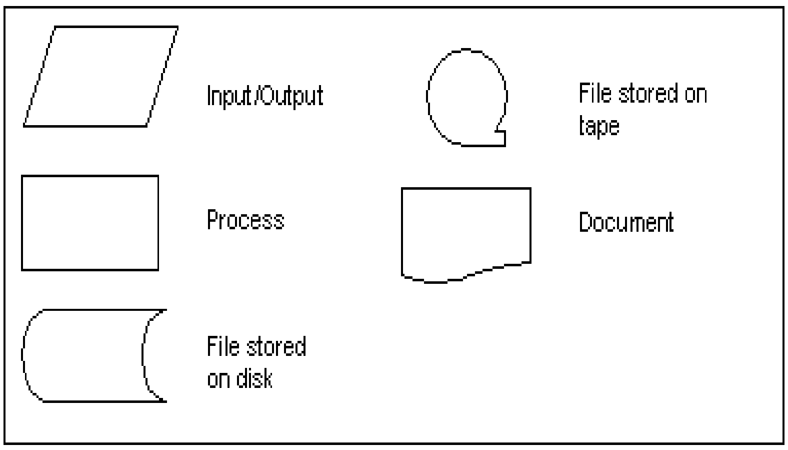
Baselines are an important part of the Systems Development Life Cycle (SDLC). These baselines are established after four of the five phases of the SDLC and are critical to the iterative nature of the model.. Each baseline is considered as a milestone in the SDLC.

- Functional Baseline: established after the conceptual design phase.
- Allocated Baseline: established after the preliminary design phase.
- Product Baseline: established after the detail design and development phase.
- Updated Product Baseline: established after the production construction phase.

8.8 SYSTEM FLOW CHART:

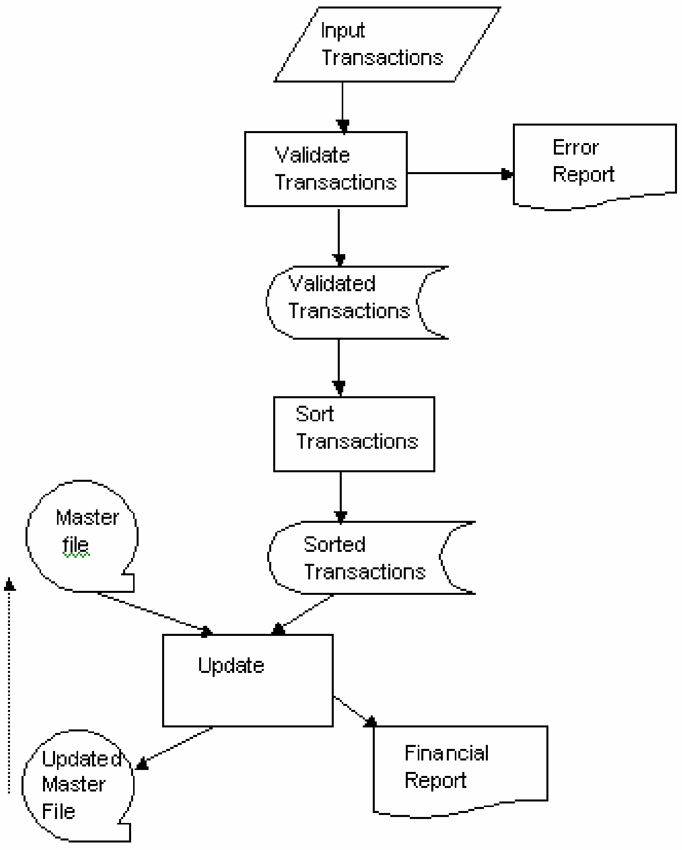
A system flowchart explains how a system works using a diagram. The diagram shows the flow of data through a system.

8.8.1 Symbols for to draw Flowchart:



The symbols are linked with directed lines (lines with arrows) showing the flow of data through the system.

8.8.2 An example of a system flowchart is shown below:

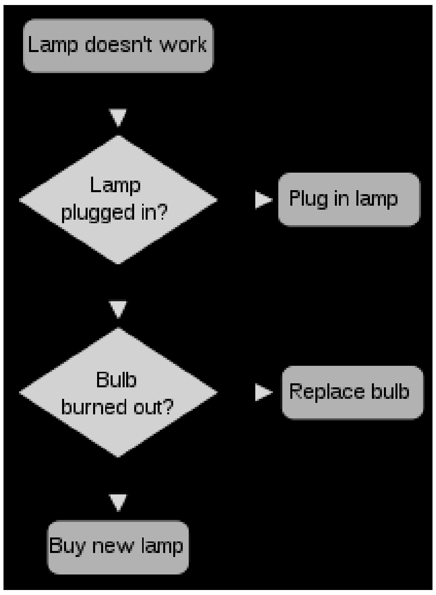


Transactions are input, validated and sorted and then used to update a master file.

Note: The arrows show the flow of data through the system. The dotted line shows that the Updated master file is then used as input for the next Update process.

A **flowchart** is a type of diagram, that represents an algorithm or process, showing the steps as boxes of various kinds, and their order by connecting these with arrows. This diagrammatic representation can give a step-by-step solution to a given problem. Data is represented in these boxes, and arrows connecting them represent flow / direction of flow of data. Flowcharts are used in analyzing, designing, documenting or managing a process or program in various fields.

Example:



8.8.3 Flowchart Symbols:

A typical flowchart from older to computer science textbooks may have the following kinds of symbols:

Start and end symbols

Represented as circles, ovals or rounded rectangles, usually containing the word "Start" or "End", or another phrase signalling the start or end of a process, such as "submit enquiry" or "receive product".

Arrows

Showing what's called "flow of control" in computer science. An arrow coming from one symbol and ending at another symbol represents that control passes to the symbol the arrow points to.

Processing steps

Represented as rectangles. Examples: "Add 1 to X"; "replace identified part"; "save changes" or similar.

Input/Output

Represented as a parallelogram.
Examples: Get X from the user; display X.

8.9 STRUCTURE CHART:

A **Structure Chart** (SC) in software engineering and organizational theory is a chart, which shows the breakdown of the configuration system to the lowest manageable levels.

This chart is used in structured programming to arrange the program modules in a tree structure. Each module is represented by a box, which contains the module's name. The tree structure visualizes the relationships between the modules.

Overview:

A structure chart is a top-down modular design tool, constructed of squares representing the different modules in the system, and lines that connect them. The lines represent the connection and or ownership between activities and sub activities as they are used in organization charts.

In structured analysis structure charts, according to Wolber (2009), "are used to specify the high-level design, or architecture, of a computer program. As a design tool, they aid the programmer in dividing and conquering a large software problem, that is, recursively breaking a problem down into parts that are small enough to be understood by a human brain. The process is called top-down design, or functional decomposition. Programmers use a structure chart to build a program in a manner similar to how an architect uses a blueprint to build a house. In the design stage, the chart is drawn and used as a way for the client and the various software designers to communicate. During the actual building of the program (implementation), the chart is continually referred to as the master-plan".

A structure chart is also used to diagram associated elements that comprise a run stream or thread. It is often developed as a hierarchical diagram, but other representations are allowable. The representation must describe the breakdown of the configuration system into subsystems and the lowest manageable level. An accurate and complete structure chart is the key to the determination of the configuration items, and a visual representation of the configuration system and the internal interfaces among its CIs. During the configuration control process, the structure chart is used to identify CIs and their associated artifacts that a proposed change may impact.

8.9.1 Applications of Structure Chart:

Use a Structure Chart to illustrate the high level overview of software structure. Structure Charts do not show module internals. Use a method, such as Pseudo code or Structured English, to show the detailed internals of modules.

Following are few advantage points:

-Representing Sequence, Repetition, and Condition on a Structure Chart

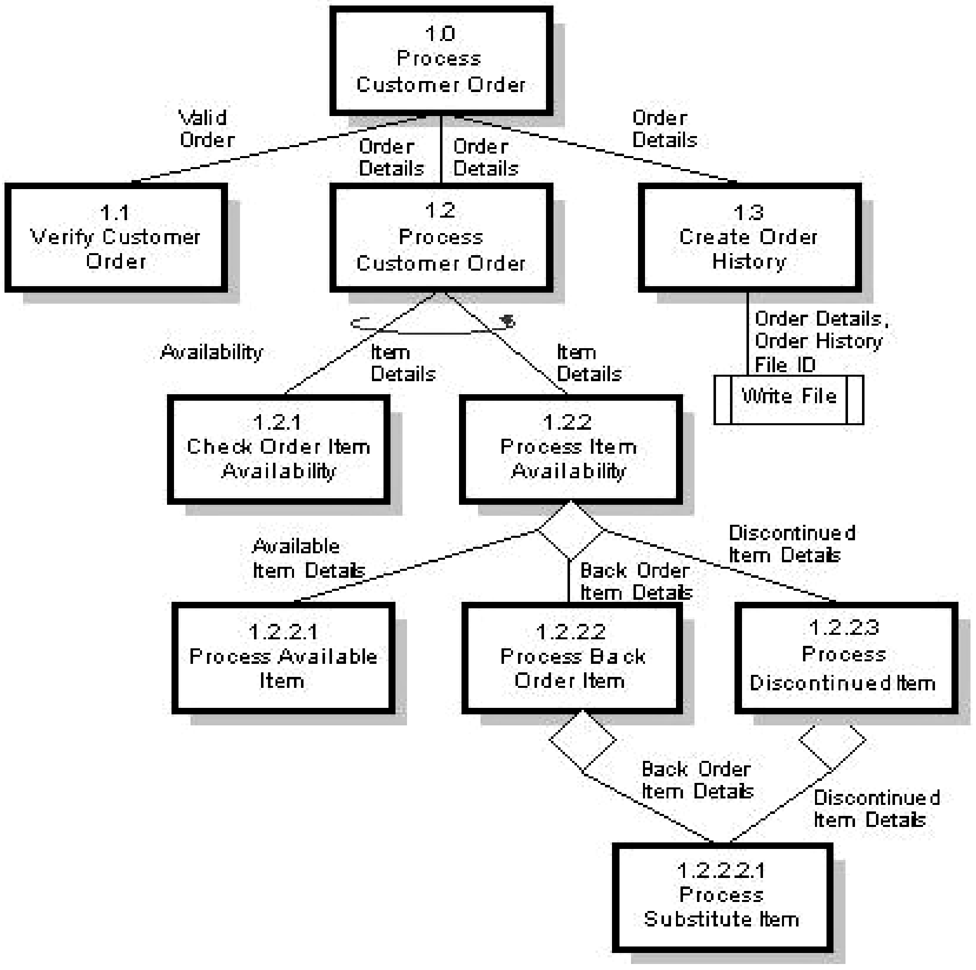
-Modules on a Structure Chart

-Interrelationships among Modules

-Information Transfers

-Reducing Clutter on a Structure Chart.

8.9.2 Example: The example Structure Chart illustrates the structure of the modules to **process a customer order**.


1.0 Process Customer Order
Valid Order
Order Details
Order Details
Order Details
1.1 Verify Customer Order
1.2 Process Customer Order
1.3 Create Order History
Order Details, Order History File ID
Write File
Availability
Item Details
Item Details
1.2.1 Check Order Item Availability
1.2.2 Process Item Availability
Available Item Details
Back Order Item Details
Discontinued Item Details
1.2.2.1 Process Available Item
1.2.2.2 Process Back Order Item
1.2.2.3 Process Discontinued Item
Back Order Item Details
Discontinued Item Details
1.2.2.2.1 Process Substitute Item

8.10 TRANSACTIONAL ANALYSIS:

Transactional Analysis is a theory developed by Dr. Eric Berne in the 1950s. **Transactional analysis**, commonly known as **TA** to its adherents, is an integrative approach to the theory of psychology and psychotherapy.

Transactional analysis can serve as a sophisticated, elegant, and effective system on which to base the practical activities of professionals in psychotherapy, counselling, education, and organizational consultation.

It is a sophisticated theory of personality, motivation, and problem solving that can be of great use to psychotherapists, counsellors, educators, and business consultants.

Transactional analysis can be divided into five theoretical and practical conceptual clusters. These five clusters enjoy varying degrees of recognition within the behavioural sciences. They are listed below along with (between quotes) concepts that parallel them in the behavioural sciences.

1. The Strokes Cluster. This cluster finds correlates in existing theories of "attachment," "intimacy," "warmth," "tender loving care," "need to belong," "contact," "closeness," "relationships," "social support," and "love."

2. The OK Cluster. This cluster finds correlates in existing theories of "positive psychology," "flow," "human potential," "resiliency," "excellence," "optimism," "subjective well-being," "positive self-concept," "spontaneous healing," "nature's helping hand," "vis medicatrix naturae" (the healing power of nature), and "the healing power of the mind."

3. The Script and Games Cluster. This cluster finds correlates in existing theories of "narratives," "maladaptive schemas," "self-narratives," "story schemas," "story grammars," "personal myths," "personal event memories," "self-defining memories," "nuclear scenes," "gendered narratives," "narrative coherence," "narrative complexity," "core self-beliefs," and "self-concept."

4. The Ego States and Transactions Cluster. The idea of three egos states and the transactional interactions between them are the most distinctive feature of transactional analysis and yet have the least amount of resonance in the literature. However, the utility of this concept is the principal reason why people become interested and maintain their interest in transactional analysis.

5. The Transactional Analysis Theory of Change Cluster. Transactional analysis is essentially a cognitive-behavioural theory of personality and change that nevertheless retains an interest in the psychodynamic aspect of the personality.

Transactional Analysis is a contractual approach. A contract is "an explicit bilateral commitment to a well-defined course of action" Berne E. (1966). This means that all parties need to agree:

- why they want to do something
- with whom
- what they are going to do
- by when
- Any fees, payment or exchanges there will be..

The fact that these different states exist is taken as being responsible for the positive or negative outcomes to conversations. Berne showed the transactional stimulus and response through the use of a simple diagram showing parent (P), adult (A), child and (C), ego states and the transactional links between them.

P P
A A
C C

8.10.1 The three ego states presented by Berne are:

Parent

The parent ego state is characterized by the need to establish standards, direct others, in still values and criticize. There are two recognized sub-groups of this ego state, being controlling parents who show signs of being authoritarian, controlling and negative, and nurturing parents who tend to be positive and supportive, but who can become suffocating.

Adult

The adult ego state is characterized by the ability to act in a detached and rational manner, logically as a decision maker utilizing information to its maximum. The archetypal example of this ego state might be Mr Spock!

Child

The child ego state is characterized by a greater demonstration of emotion, either positive or negative. Once again, as with the parent, there are sub-groups of this ego state, in this case three. The first is the natural child state, with uninhibited actions, which might include energy and raw enthusiasm, to curiosity and fear. It is essentially self- centred. The adapted child state is a state where emotions are still strong, but there is some attempt to control, ending in compliant or withdrawn behaviours. Finally, the 'little professor' is a child ego state that shows emerging adult traits, and a greater ability to constrain emotions.

Transactions can be brief, can involve no verbal content at all (looking at some-one across a room), or can be long and involved. However, Berne believed that there were four basic types of transaction:

8.10.2 The four basic types of transaction:

Complementary

A transaction where the ego states complement each other, resulting in a positive exchange. This might include two teachers discussing some assessment data in order to solve a problem where they are both inhabiting the adult ego state.

Duplex

This is a transaction that can appear simple, but entails two levels of communication, one often implicit. At a social level, the transaction might be adult to adult, but at a psychological level it might be child to child as a hidden competitive communication.

Angular

Here, the stimulation appears to be aimed at one ego state, but covertly is actually aimed at another, such as the use of sarcasm. This may then lead to a different ego state response from that which might be expected.

Crossed

Here, the parent acts as a controlling parent, but in aiming the stimulus at the child ego state, a response from the adult ego state, although perhaps perfectly reasonable but unexpected, brings conflict.

As a result, where there are crossed transactions, there is a high possibility of a negative development to a conversation, often resulting in confrontation or bad feeling?

8.10.3 Example:

Transactional Analysis in the classroom:

Within any classroom there is a constant dynamic transactional process developing. How this is man-aged can have important ramifications for both short and long term relationships between staff and students.

If we take as a starting point the more traditional style of relationship between teacher and student, this will often occur as a parental stimulus directed at a child, expecting a child reaction. Hence, this translates to a simple transaction such as that below (shown by the solid lines).

However, as all teachers know, students at secondary level are beginning to re-establish their boundaries as people and are becoming increasingly independent. As a result, it is increasingly likely that as the children become older, a parental stimulus directed at a child ego state will result in an adult to adult response.

Even though the response is perfectly reasonable, and indeed would be sought in most circumstances, in this case it leads to a crossed transaction and the potential for a negative conversation.

8.11 SUMMARY

This chapter is based on System design and their different design models like flow chart, UML diagram, structure chart, activity diagram. These diagram helps to represent flow and working of data.

❖❖❖❖

9

SOFTWARE DESIGN AND DOCUMENTATION TOOLS

Unit Structure

9.1 INTRODUCTION:

Software documentation or source code documentation is written text that accompanies computer software. It either explains how it operates or how to use it, and may mean different things to people in different roles.

9.2 PEOPLE AND SOFTWARE:

Documentation is an important part of software engineering. Types of documentation include**:**

1. Requirements - Statements that identify attribute capabilities, characteristics, or qualities of a system. This is the foundation for what shall be or has been implemented.
2. Architecture/Design - Overview of software. Includes relations to an environment and construction principles to be used in design of software components.
3. Technical - Documentation of code, algorithms, interfaces, and APIs.
4. End User - Manuals for the end-user, system administrators and support staff.

5. Marketing - How to market the product and analysis of the market demand.

9.3 REQUIREMENTS DOCUMENTATION:

Requirements documentation is the description of what particular software does or shall do. It is used throughout development to communicate what the software does or shall do. It is also used as an agreement or as the foundation for agreement on what the software shall do. Requirements are produced and consumed by everyone involved in the production of software: end users, customers, product managers, project anagers, sales, marketing, software architects, usability experts, interaction designers, developers, and testers, to name a few. Thus, requirements documentation has many different purposes.

The need for requirements documentation is typically related to the complexity of the product, the impact of the product, and the life expectancy of the software. If the software is very complex or developed by many people (e.g., mobile phone software), requirements can help to better communicate what to achieve. If the software is safety-critical and can have negative impact on human life (e.g., nuclear power systems, medical equipment), more formal requirements documentation is often required. If the software is expected to live for only a month or two (e.g., very small mobile phone applications developed specifically for a certain campaign) very little requirements documentation may be needed. If the software is a first release that is later built upon, requirements documentation is very helpful when managing the change of the software and verifying that nothing has been broken in the software when it is modified.

9.4 ARCHITECTURE/DESIGN DOCUMENTATION:

Architecture documentation is a special breed of design document. In a way, architecture documents are third derivative from the code (design document being second derivative, and code documents being first). Very little in the architecture documents is specific to the code itself. These documents do not describe how to program a particular routine, or even why that particular routine exists in the form that it does, but instead merely lays out the general requirements that would motivate the existence of such a routine. A good architecture document is short on details but thick on explanation. It may suggest approaches for lower level design, but leave the actual exploration trade studies to other documents.

A very important part of the design document in enterprise software development is the Database Design Document (DDD). It contains Conceptual, Logical, and Physical Design Elements. The DDD includes the formal information that the people who interact with the database need. The purpose of preparing it is to create a common source to be used by all players within the scene. The potential users are:

- Database Designer
- Database Developer
- Database Administrator
- Application Designer
- Application Developer

When talking about Relational Database Systems, the document should include following parts:

- Entity - Relationship Schema, including following information and their clear definitions:
- Entity Sets and their attributes
- Relationships and their attributes
- Candidate keys for each entity set
- Attribute and Tuple based constraints
- Relational Schema, including following information:
    - Tables, Attributes, and their properties
    - Views
    - Constraints such as primary keys, foreign keys,
    - Cardinality of referential constraints
    - Cascading Policy for referential constraints
    - Primary keys

It is very important to include all information that is to be used by all actors in the scene. It is also very important to update the documents as any change occurs in the database as well.

9.5 TECHNICAL DOCUMENTATION:

This is what most programmers mean when using the term software documentation. When creating software, code alone is insufficient. There must be some text along with it to describe various aspects of its intended operation. It is important for the code documents to be thorough, but not so verbose that it becomes difficult to maintain them. Several How-to and overview documentation are found specific to the software application or software product being documented by API Writers. This documentation may be used by developers, testers and also the end customers or clients using this software application.

Many programmers really like the idea of auto-generating documentation for various reasons. For example, because it is extracted from the source code itself (for example, through comments), the programmer can write it while referring to the code, and use the same tools used to create the source code to make the documentation. This makes it much easier to keep the documentation up-to-date.

9.6 USER DOCUMENTATION:

Unlike code documents, user documents are usually far more diverse with respect to the source code of the program, and instead simply describe how it is used.

In the case of a software library, the code documents and user documents could be effectively equivalent and are worth conjoining, but for a general application this is not often true. On the other hand, the Lisp machine grew out of a tradition in which every piece of code had an attached documentation string. In combination with strong search capabilities (based on a Unix-like apropos command), and online sources, Lisp users could look up documentation prepared by these API Writers and paste the associated function directly into their own code. This level of ease of use is unheard of in putatively more modern systems.

Typically, the user documentation describes each feature of the program, and assists the user in realizing these features. A good user document can also go so far as to provide thorough troubleshooting assistance. It is very important for user documents to not be confusing, and for them to be up to date. User documents need not be organized in any particular way, but it is very important for them to have a thorough index. Consistency and simplicity are also very valuable. User documentation is considered to constitute a contract specifying what the software will do. API Writers are very well accomplished towards writing good user documents as they would be well aware of the software architecture and programming techniques used. See also Technical Writing.

There are three broad ways in which user documentation can be organized:

1. **Tutorial:** A tutorial approach is considered the most useful for a new user, in which they are guided through each step of accomplishing particular tasks.

2. Thematic: A thematic approach, where chapters or sections concentrate on one particular area of interest, is of more general use to an intermediate user. Some authors prefer to convey their ideas through a knowledge based article to facilitating the user needs. This approach is usually practiced by a dynamic industry, such as Information technology, where the user population is largely correlated with the troubleshooting demands.
3. List or Reference: The final type of organizing principle is one in which commands or tasks are simply listed alphabetically or logically grouped, often via cross-referenced indexes. This latter approach is of greater use to advanced users who know exactly what sort of information they are looking for.

9.7 MARKETING DOCUMENTATION:

For many applications it is necessary to have some promotional materials to encourage casual observers to spend more time learning about the product. This form of documentation has three purposes:-

1. To excite the potential user about the product and in still in them a desire for becoming more involved with it.
2. To inform them about what exactly the product does, so that their expectations are in line with what they will be receiving.
3. To explain the position of this product with respect to other alternatives.
4. To completely shroud the function of the product in mystery.

9.8 HIPO CHART:

HIPO for hierarchical input process output is a "popular 1970s systems analysis design aid and documentation technique for representing the modules of a system as a hierarchy and for documenting each module.

It was used to develop requirements, construct the design, and support implementation of an expert system to demonstrate automated rendezvous. Verification was then conducted systematically because of the method of design and implementation.

The overall design of the system is documented using HIPO charts or structure charts. The structure chart is similar in appearance to an organizational chart, but has been modified to show additional detail. Structure charts can be used to display several types of information, but are used most commonly to diagram either data structures or code structures.

Why we use HIPO chart?

The HIPO (Hierarchy plus Input-Process-Output) technique is a tool for planning and/or documenting a computer program. A HIPO model consists of a hierarchy chart that graphically represents the program's control structure and a set of IPO (Input-Process-Output) charts that describe the inputs to, the outputs from, and the functions (or processes) performed by each module on the hierarchy chart.

Advantages of HIPO Chart:

Using the HIPO technique, designers can evaluate and refine a program's design, and correct flaws prior to implementation. Given the graphic nature of HIPO, users and managers can easily follow a program's structure. The hierarchy chart serves as a useful planning and visualization document for managing the program development process. The IPO charts define for the programmer each module's inputs, outputs, and algorithms.

Limitation of HIPO Chart:

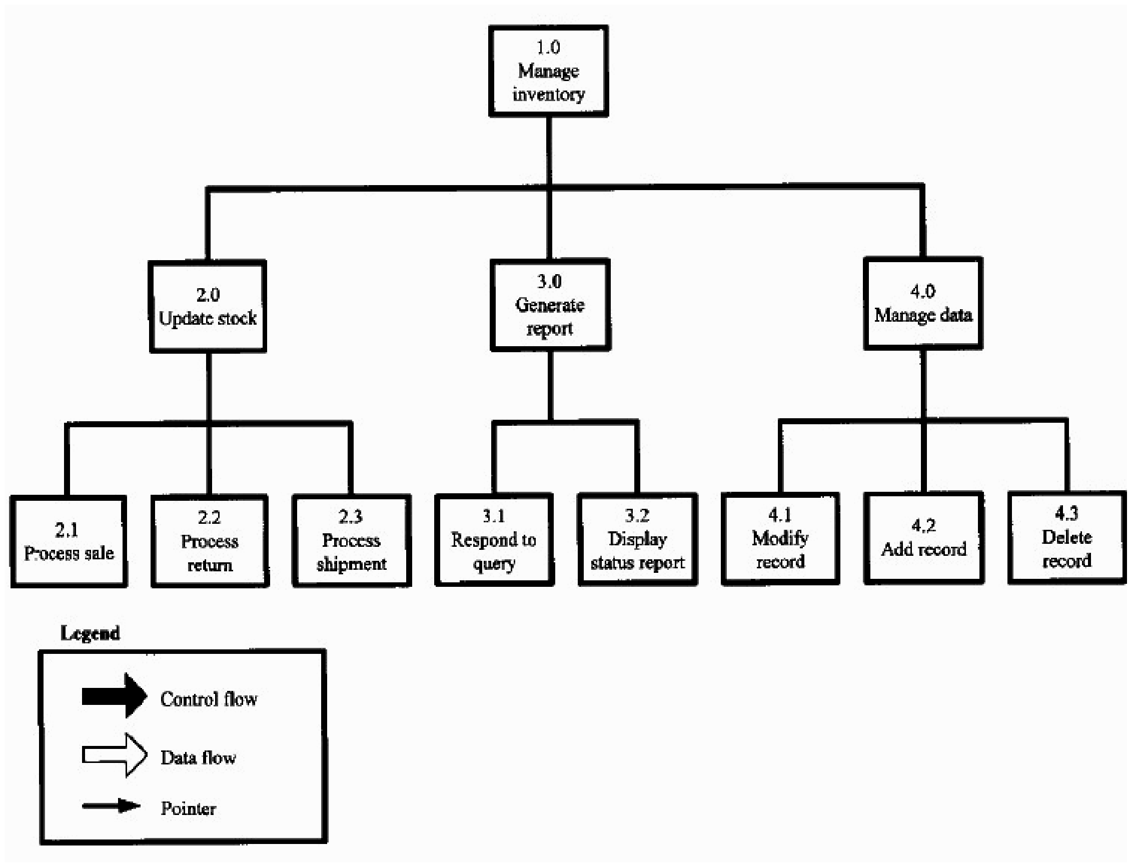
-HIPO provides valuable long-term documentation. However, the "text plus flowchart" nature of the IPO charts makes them difficult to maintain, so the documentation often does not represent the current state of the program.

-By its very nature, the HIPO technique is best used to plan and/or document a hierarchically structured program.

Example: Set of Tasks to Be Performed by an Interactive Inventory Program:-

1.0 Manage inventory
2.0 Update stock
2.1 Process sale
2.2 Process return
2.3 Process shipment
3.0 Generate report
3.1 Respond to query
3.2 Display status report
4.0 Maintain inventory data
4.1 Modify record
4.2 Add record
4.3 Delete record

Diagrammatically presentation: A hierarchy chart for an interactive inventory control program.



9.9 WARNIER-ORR DIAGRAM:

A **Warnier/Orr diagram** (also known as a logical construction of a program/system) is a kind of hierarchical flowchart that allow the description of the organization of data and procedures.

Warnier/Orr diagrams show the processes and sequences in which they are performed. Each process is defined in a hierarchical manner i.e. it consists of sets of subprocesses, that define it. At each level, the process is shown in bracket that groups its components.

Since a process can have many different subprocesses, Warnier/Orr diagram uses a set of brackets to show each level of the system. Critical factors in s/w definition and development are iteration or repetition and alteration. Warnier/Orr diagrams show this very well.

To develop a Warnier/Orr diagram, the analyst works backwards, starting with systems output and using output oriented analysis. On paper, the development moves from right to left. First, the intended output or results of the processing are defined. At the next level, shown by inclusion with a bracket, the steps needed to produce the output are defined. Each step in turn is further defined. Additional brackets group the processes required to produce the result on the next level.

Construct the Warnier-orr diagram:

There are four basic constructs used on Warnier/Orr diagrams: hierarchy, sequence, repetition, and alternation. There are also two slightly more advanced concepts that are occasionally needed: concurrency and recursion.

Hierarchy

Hierarchy is the most fundamental of all of the Warnier/Orr constructs. It is simply a nested group of sets and subsets shown as a set of nested brackets. Each bracket on the diagram (depending on how you represent it, the character is usually more like a brace "{" than a bracket "[", but we call them "brackets") represents one level of hierarchy. The hierarchy or structure that is represented on the diagram can show the organization of data or processing. However, both data and processing are never shown on the same diagram.

Sequence

Sequence is the simplest structure to show on a Warnier/Orr diagram. Within one level of hierarchy, the features listed are shown in the order in which they occur. In other words, on the Fig. 6 the step listed first is the first that will be executed (if the diagram reflects a process), while the step listed last is the last that will be executed. Similarly with data, the data field listed first is the first that is encountered when looking at the data, the data field listed last is the final one encountered.

Repetition

Repetition is the representation of a classic "loop" in programming terms. It occurs whenever the same set of data occurs over and over again (for a data structure) or whenever the same group of actions is to occur over and over again (for a processing structure).

lternation

Alternation, or selection, is the traditional "decision" process whereby a determination is made to execute one process or another. It is indicated as a relationship between two subsets.

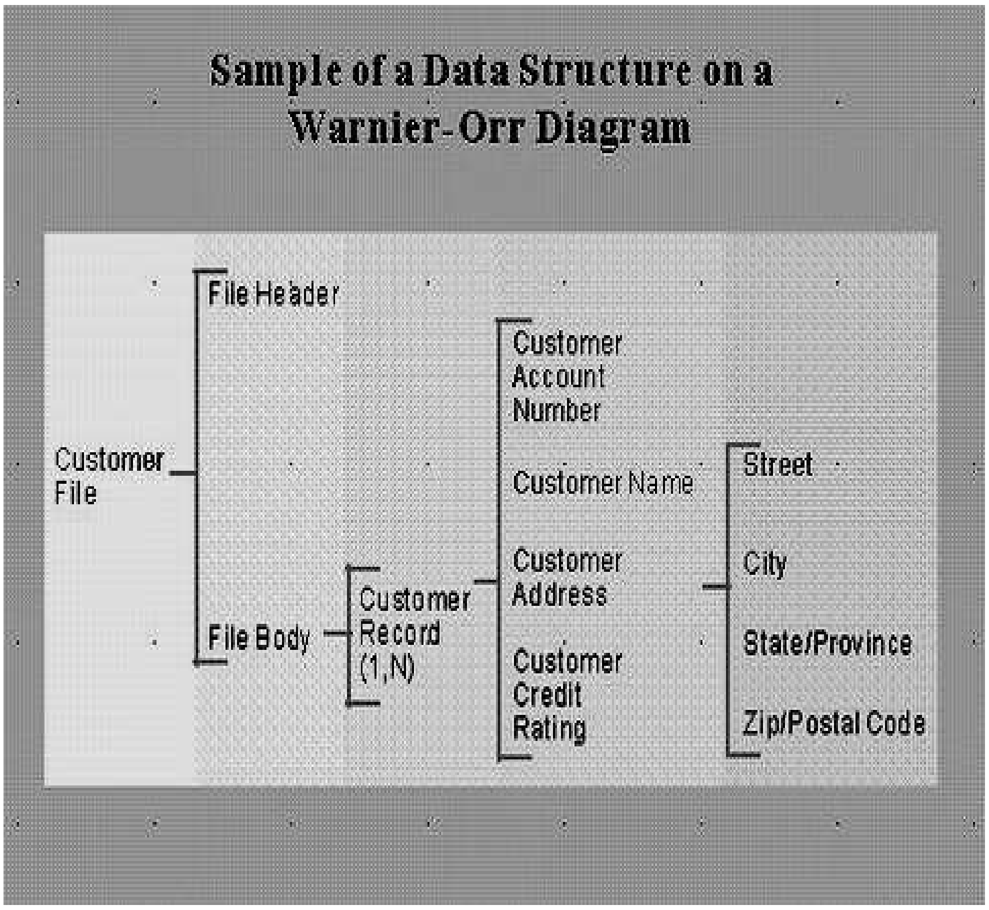
Concurrency

Concurrency is one of the two advanced constructs used in the methodology. It is used whenever sequence is unimportant.

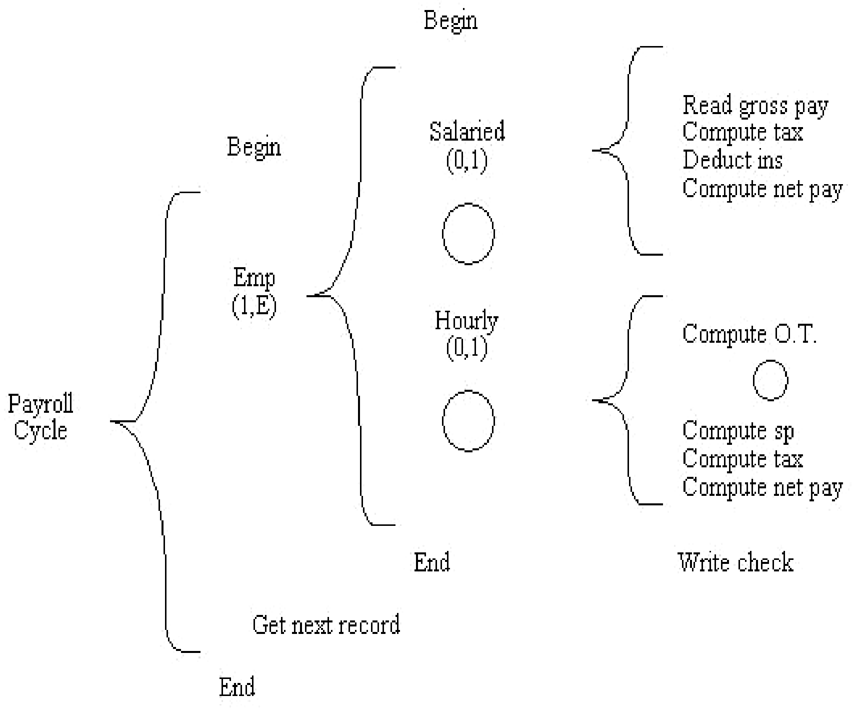
Recursion

Recursion is the least used of the constructs. It is used to indicate that a set contains an earlier or a less ordered version of itself.

Example: Warnier-orr diagram for Data Structure.



Example: Warnier Orr diagram for Payroll cycle.



Example:

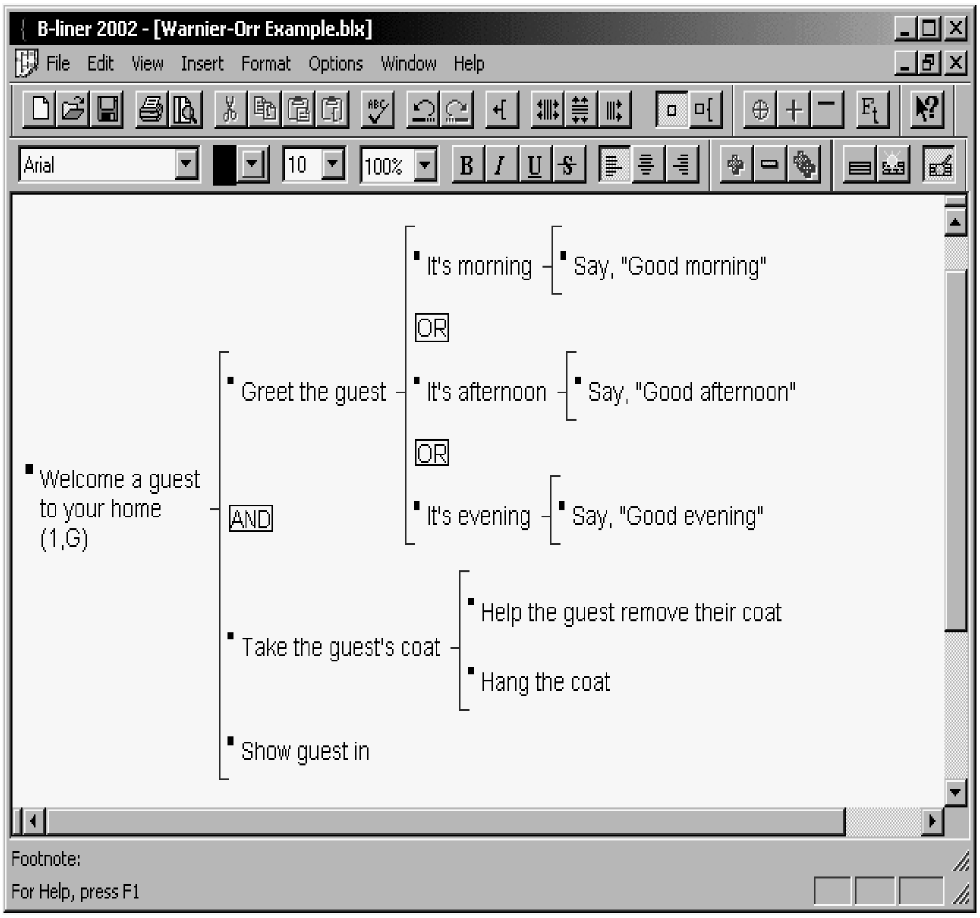
A **Warnier/Orr** diagram is a style of diagram which is extremely useful for describing complex processes (e.g. computer programs, business processes, instructions) and objects (e.g. data structures, documents, parts explosions). Warnier/Orr diagrams are elegant, easy to understand and easy to create. When you interpret one of B-liner's diagrams as a Warnier/Orr diagram, you give a simple, yet formal meaning to the elements of the diagram.

The following is a quick description of the main elements of a Warnier/Orr diagram.

Bracket:	A bracket encloses a level of decomposition in a diagram. It reveals what something "consists of" at the next level of detail.
Sequence:	The sequence of events is defined by the top-to-bottom order in a diagram. That is, an event occurs after everything above it in a diagram, but before anything below it.
OR:	You represent choice in a diagram by placing an "OR" operator between the items of a choice. The "OR" operator looks either like or .
AND:	You represent concurrency in a diagram by placing an "AND" operator between the concurrent actions. The "AND" operator looks either like or .

Repetition: To show that an action repeats (loops), you simply put the number of repetitions of the action in parentheses below the action.

The diagram below illustrates the use of these constructs to describe a simple process.



You could read the above diagram like this:

"Welcoming a guest to your home (from 1 to many times) consists of greeting the guest and taking the guest's coat at the same time, then showing the guest in. Greeting a guest consists of saying "Good morning" if it's morning, or saying "Good afternoon" if it's afternoon, or saying "Good evening" if it's evening. Taking the guest's coat consists of helping the guest remove their coat, then hanging the coat up."

As you can see, the diagram is much easier to understand than the description.

9.10 SUMMARY

This chapter used different software tools to find out working style, process flow, data determination with the help of some latest tools.

Questions:

1. Explain Requirements documentation in detail?

Ans: Refer 9.3

2. Explain Architecture/Design documentation in detail?

Ans: refer 9.4

3. Explain Technical documentation in detail?

Ans: refer 9.5

4. Explain Hipo chart?

Ans: refer 9.8

❖❖❖❖

10

DESIGNING INPUT, OUTPUT & USER INTERFACE

Unit Structure

10.1 INTRODUCTION:

Output is what the customer is buying when he or she pay for a development of project. Inputs, databases, and processes are present to provide output.

A data input specification is a detailed description of the individual fields (data elements) on an input document together with their characteristics. In this chapter we will learn about Input design, Output design and User Interface.

10.2 OUTPUT DESIGN:

Output is the most important task of any system. These guidelines apply for the most part to both paper and screen outputs. Output design is often discussed before other feature of design because, from the customer's point of view, the output is the system. Output is what the customer is buying when he or she pay for a development of project. Inputs, databases, and processes are present to provide output. Problems often associated with business information output are information hold-up, information (data) overload, paper domination, extreme distribution, and no tailoring.

For example:

Mainframe printers: high volume, high speed, located in the data centre Remote site printers: medium speed, close to end user.

COM is Computer Output Microfilm. It is more compressed than traditional output and may be produced as fast as non-impact printer output.

- Turnaround documents trim down the cost of internal information processing by reducing both data entry and associated errors.
- Periodic reports have set frequencies such as daily or weekly; ad hoc reports are produced at irregular intervals.
- Detail and summary reports differ in the former support day-to-day operation of the business while the latter include statistics and ratios used by managers to consider the health of operations.
- Page breaks and control breaks allow for abstract totals on key fields. Report requirements documents include general report information and field specifications; print layout sheets present a picture of what the report will actually look like.
- Page decoupling is the separation of pages into cohesive groups.

Two ways to create output for strategic purposes are

(1) Make it compatible with processes outside the immediate scope of the system

(2) Turn action documents into turnaround documents.

People often receive reports they do not require because the number of reports received is perceived as a measure of power. Fields on a report should be selected carefully to provide organized reports, facilitate 80-column remote printing, and reduce information (data) overload.

The types of fields which should be considered for business output are: key fields for access to information, fields for control breaks, fields that change, and exception fields.

Output may be designed to aid future change by stressing formless reports, defining field size for future growth, making field constants into variables, and leaving room on review reports for added ratios and statistics.

Output can now be more easily tailored to the needs of individual users because inquiry-based systems allow users themselves to generate ad hoc reports. An output intermediary can restrict access to key information and avoid illegal access. An information clearinghouse (or information centre) is a service centre that provides consultation, assistance, and documentation to encourage end-user development and use of applications. The specifications essential to describe the output of a system are: data flow diagrams, data flow specifications, data structure specifications, and data element specifications.

- Output Documents
- Printed Reports
- External Reports: for use or distribution outside the organization; often on pre-printed forms.
- Internal Reports: for use within the organization; not as "pretty", stock paper, greenbar, etc.
- Periodic Reports: produced with a set frequency (daily, weekly, monthly, every fifth Tuesday, etc.)
- Ad-Hoc (On Demand) Reports: unbalanced interval; produced upon user demand.
- Detail Reports: one line per transaction.

  Review Reports: an overview.
- Exception Reports: only shows errors, problems, out-of-range values, or unexpected conditions or events.

10.3 INPUT DESIGN

A source document differs from a turnaround document in that the former holds data that revolutionize the status of a resource while the latter is a machine readable document. Transaction throughput is the number of error-free transactions entered during a specified time period. A document should be concise because longer documents contain more data and so take longer to enter and have a greater chance of data entry errors.

Numeric coding substitutes numbers for character data (e.g., 1=male, 2=female); mnemonic coding represents data in a form that is easier for the user to understand and remember. (e.g., M=male, F=female). The more quickly an error is detected, the nearer the error is to the person who generated it and so the error is more easily corrected. An example of an illogical combination in a payroll system would be an option to eliminate federal tax withholding.

By "multiple levels" of messages, I mean allowing the user to obtain more detailed explanations of an error by using a help option, but not forcing a long-lasting message on a user who does not want it. An error suspense record would include the following fields: data entry operator identification, transaction entry date, transaction entry time, transaction type, transaction image, fields in error, error codes, date transaction re-entered successfully.

- A data input specification is a detailed description of the individual fields (data elements) on an input document together with their characteristics (i.e., type and length).
- Be specific and precise, not general, ambiguous, or vague. (BAD: Syntax error, Invalid entry, General Failure)
- Don't JUST say what's wrong---- Be constructive; propose what needs to be done to correct the error condition.
- Be positive; Avoid condemnation. Possibly even to the point of avoiding pejorative terms such as "invalid" "illegal" or "bad."
- Be user-centric and attempt to convey to the user that he or she is in control by replacing imperatives such as "Enter date" with wording such as "Ready for date."
- Consider multiple message levels: the initial or default error message can be brief but allow the user some mechanism to request additional information.
- Consistency in terminology and wording.
    i. Place error messages in the same place on the screen
    ii. Use consistent display characteristics (blinking, colour, beeping, etc.)

10.4 USER INTERFACE

i. The primary differences between an inter active and batch environment are:

- interactive processing is done during the organization's prime work hours
- interactive systems usually have multiple, simultaneous users
- the experience level of users runs from novice to highly experienced
- developers must be good communicators because of the need to design systems with error messages, help text, and requests for user responses.

ii. The seven step path that grades the structure of an interactive system is
    a. Greeting screen (e.g., company logo)

   b. Password screen -- to prevent unauthorized use
   c. Main menu -- allow choice of several available applications
   d. Intermediate menus -- further delineate choice of functions
   e. Function screens -- updating or deleting records
   f. Help screens -- how to perform a task
   g. Escape options -- from a particular screen or the application

iii. An intermediate menu and a function screen differ in that the former provides choices from a set of related operations while the latter provides the ability to perform tasks such as updates or deletes.

iv. The difference between inquiry and command language dialogue modes is that the former asks the user to provide a response to a simple question (e.g., "Do you really want to delete this file?") where the latter requires that the user know what he or she wants to do next (e.g., MS-DOS C:> prompt; VAX/VMS $ prompt; Unix shell prompt). GUI Interface (Windows, Macintosh) provide Dialog Boxes to prompt user to input required information/parameters.

v. Directions for designing form-filling screens:

   a) Fields on the screen should be in the same sequence as on the source document.
   b) Use cuing to provide the user with information such as field formats (e.g., dates)
   c) Provide default values.
   d) Edit all entered fields for transaction errors.
   e) Move the cursor automatically to the next entry field
   f) Allow entry to be free-form (e.g., do not make the user enter leading zeroes)

Consider having all entries made at the same position on the screen.

vi. A default value is a value automatically supplied by the application when the user leaves a field blank. For example, at SXU the screen on which student names and addresses are entered has a default value of "IL" for State since the majority of students have addresses in Illinois. At one time "312" was a default value for Area Code, but with the additional Area Codes now in use (312, 773, 708, 630, 847) providing a default value for this field is no longer as useful.

vii. The eight parts of an interactive screen menu are:

0. Locator -- what application the user is currently in
1. Menu ID -- allows the more experienced user access without going through the entire menu tree.
2. Title
3. User instructions
4. Menu list
5. Escape option
6. User response area
7. System messages (e.g., error messages)

viii. Highlighting should be used for gaining attention and so should be limited to critical information, unusual values, high priority messages, or items that must be changed.

ix. Potential problems associated with the overuse of color are:

- Colors have different meanings to different people and in different cultures.
- A certain percentage of the population is known to have color vision deficiency.
- Some color combinations may be disruptive.

x. Information density is important because density that is too high makes it more difficult to discern the information presented on a screen, especially for novice users.

xi. Rules for defining message content include:

- Use active voice.
- Use short, simple sentences.
- Use affirmative statements.
- Avoid hyphenation and unnecessary punctuation.
- Separate text paragraphs with at least one blank line.
- Keep field width within 40 characters for easy reading.
- Avoid word contractions and abbreviations.
- Use non threatening language.
- Avoid godlike language.
- Do not patronize.
- Use mixed case (upper and lower case) letters.
- Use humour carefully.

xii. Symmetry is important to screen design because it is aesthetically pleasing and thus more comforting.

xiii. Input verification is asking the user to confirm his or her most recent input (e.g., "Are you sure you want to delete this file?")

xiv. Adaptive models are useful because they adapt to the user's experience level as he or she moves from novice to experienced over time as experience with the system grows.

xv. "Within User" sources of variation include: warm up, fatigue, boredom, environmental conditions, and extraneous events.

xvi. The elements of the adaptive model are:

- Triggering question to determine user experience level
- Differentiation among user experience
- Alternative processing paths based on user level
- Transition of casual user to experienced processing path
- Transition of novice user to experienced processing path
- Allowing the user to move to an easier processing path

xvii. Interactive tasks can be designed for closure by providing the user with feedback indicating that a task has been completed.

xviii. Internal locus of control is making users feel that they are in control of the system, rather than that the system is in control of them.

xix. Examples of distracting use of surprise are:

- Highlighting
- Input verification
- Flashing messages
- Auditory messages

xx. Losing the interactive user can be avoided by using short menu paths and "You are here" prompts.

xxi. Some common user shortcuts are: direct menu access, function keys, and shortened response time.

10.5 GOLDEN RULES OF INTERFACE DESIGN:

1. Strive for consistency.
2. Enable frequent users to use shortcuts.
3. Offer informative feedback.
4. Design dialogs to yield closure.
5. Offer error prevention and simple error handling.
6. Permit easy reversal of actions.

7. Support internal locus of control.
8. Reduce short-term memory load.

10.6 SUMMARY:

In above chapter, we learn the concept of Output Design (Output is what the customer is buying when he or she pay for a development of project), Input Design, User Interface and rules of Interface design.

Questions:

1. Explain Golden rules of Interface Design?

Ans: refer 10.5

2. Explain Input design, output design and user interface in detail?

Ans: refer 10.2, 10.3 and 10.4

❖❖❖❖

11

SOFTWARE TESTING STRATEGY

Unit Structure

11.1 INTRODUCTION:

A strategy for software testing must accommodate low-level tests that are necessary to verify that a small source code segment has been correctly implemented as well as high-level tests that validate major system functions against customer requirements. A strategy must provide guidance for the practitioner and a set of milestones for the manager. Because the steps of the test strategy occur at a time when dead-line pressure begins to rise, progress must be measurable and problems must surface as early as possible.

11.2 STRATEGIC APPROACH TO SOFTWARE TESTING

Testing is a set of activities that can be planned in advance and conducted systematically. For this reason a template for software testing **--** a set of steps into which we can place specific test case design techniques and testing methods **--** should be defined for the software process.

A number of software testing strategies have been proposed in the literature. All provide the software developer with a template for testing and all have the following generic characteristics.

- Testing begins at the component level and works 'outward' toward the integration of the entire computer-based system,
- Different testing techniques are appropriate at different points in time.
- Testing is conducted by the developer of the software and (for large projects) an independent test group.
- Testing and debugging are different activities, but debugging must be accommodated in any testing strategy,

A strategy for software testing must accommodate low-level tests that are necessary to verify that a small source code segment has been correctly implemented as well as high-level tests that validate major system functions against customer requirements. A strategy must provide guidance for the practitioner and a set of milestones for the manager. Because the steps of the test strategy occur at a time when dead-line pressure begins to rise, progress must be measurable and problems must surface as early as possible.

11.3 ORGANIZING FOR SOFTWARE TESTING

For every software project, there is an inherent conflict of interest that occurs as testing begins. The people who have built the software are now asked to test the software. This seems harmless in itself; after all, who knows the program better than its developers? Unfortunately, these same developers have a vested interest in demonstrating that the program is error free, that it works according to customer requirements, and that it will be completed on schedule and within budget. Each of these interests mitigate against thorough testing.

From a psychological point of view, software analysis and design (along with coding) are *constructive* tasks. The software engineer creates a computer program, its documentation, and related data structures. Like any builder, the software engineer is proud of the edifice that has been built and looks askance at anyone who attempts to tear it down. When testing commences, there is a subtle, yet definite, attempt to 'break' the thing that the software engineer has built. From the point of view of the builder, testing can be considered to be (psychologically) *destructive.*

There are often a number of misconceptions that can be erroneously inferred from the preceding discussion:

- That the developer of software should do no testing at all.
- That the software should be ‘tossed over the wall’ to strangers who will test it mercilessly,
- That tester gets involved with the project only when the testing steps are about to begin.

Each of these above statements is incorrect.

The software developer is always responsible for testing the individual units (components) of the program, ensuring that each performs the function for which it was designed. In many cases, the developer also conducts integration testing -- a testing step that leads to the construction (and test) of the complete program structure. Only after the software architecture is complete does an independent test group become involved.

The role of an *independent test group* (ITG) is to remove the inherent problems associated with letting the builder test the thing that has been built. Independent testing removes the conflict of interest that may otherwise be present. After all, personnel in the independent group team are paid to find errors.

However, the software engineer doesn’t turn the program over to ITG and walk away. The developer and the ITG work closely throughout a software project to ensure that thorough tests will be conducted: While testing is conducted, the developer must be available to correct errors that are uncovered.

The ITG is part of the software development project team in the sense that it becomes involved during the specification activity and stays involved (planning and specifying test procedures) throughout a large project. However, in many cases the ITG reports to the software quality assurance organization, thereby achieving a degree of independence that might not be possible if it were a part of the software engineering organization.

11.4 A SOFTWARE TESTING STRATEGY

The software engineering process may be viewed as the spiral illustrated in figure below. Initially, system engineering defines the role of software and leads to software requirements analysis. Where the information domain, function, behaviour, performance, constraints. And validation criteria for software are established. Moving inward along the spiral we come to design and finally to coding. To develop computer software, we spiral inward along streamlines that decrease the level of abstraction on each turn.

A strategy for software testing may also be viewed in the context of the spiral (figure above). *Unit testing* begins at the vortex of the spiral and concentrates on each unit (i.e, component) of the software as implemented in source code. Testing progresses by moving outward along the spiral to *integration testing,* where the focus is on design and the construction of the software architecture. Taking another turn outward on the spiral, we encounter *validation testing,* where requirements established as part of software requirements analysis are validated against the software that has been constructed. Finally, we arrive at *system testing,* where the software and other system elements are tested as a whole. To test computer software, we spiral out along streamlines that broaden the scope of testing with each turn.

Initially, tests focus on each component individually, ensuring that it functions properly as a unit. Hence, the name *unit testing.* Unit testing makes heavy use of white-box testing techniques, exercising specific paths in a module's control structure to ensure complete coverage and maximum error detection. Next, components must be assembled or integrated to form the complete software package. Integration testing addresses the issues associated with the dual problems of verification and program construction. Black-box test case design techniques are the most prevalent during integration, although a limited amount of white-box testing may be used to ensure coverage of major control paths. After the software has been integrated (constructed), a set of high-order tests are conducted Validation criteria (established during requirements analysis) must be tested. Validation testing provides final assurance that software meets all functional, behavioural, and performance requirement. Black box testing techniques arc used exclusively during validation.

The last high-order testing step falls outside the boundary of software engineering and into the broader context of computer system engineering. Software, once validated, must be combined with other system elements (e.g., hardware, people, and databases). System testing verifies that all elements mesh properly and that overall system function/performance is achieved.

11.5 UNIT TESTING

Unit testing focuses verification effort on the smallest unit of software design -- the software component or module. Using the component-level design description as a guide, important control paths are tested to uncover errors within the boundary of the module. The relative complexity of tests and uncovered errors is limited by the constrained scope established for unit testing. The unit test is white-box oriented, and the step can be conducted in parallel for multiple components.

11.6 INTEGRATION TESTING

A neophyte in the software world might ask a seemingly legitimate question once all modules have been unit tested: "If they all work individually, why do you doubt that they'll work when we put them together?" The problem, or course, is putting them together -- interfacing. Data can be lost across an interface; one module can have an inadvertent, adverse affect on another; sub-functions, when combined, may not produce the desired major function; individually acceptable imprecision may be magnified to unacceptable levels; global data structures can present problems. Sadly, the list goes on and on.

Integration testing is a systematic technique for constructing the program structure while at the same time conducting tests to uncover errors associated with interfacing. The objective is to take unit tested components and build a program structure that has been dictated by design.

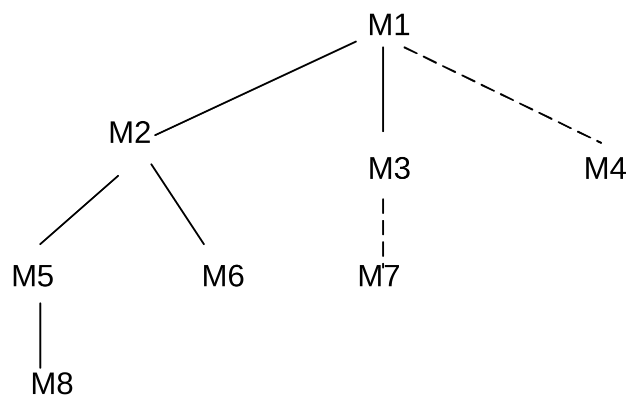
There is often a tendency to attempt non-incremental integration; that is, to construct the program using a ‘big bang’ approaches. All components are combined in advance. The entire program is tested as a whole. And chaos usually results! A set of errors is encountered. Correction is difficult because isolation or causes is complicated by the vast expanse of the entire program. Once these errors are corrected, new ones appear and the process continues in a seemingly endless loop.

Incremental integration is the antithesis of the big bang approach. The program is constructed and tested in small increments, where errors are easier to isolate and correct; interfaces are more likely to be tested completely; and a systematic test approach may be applied. In the sections that follow, a number of different incremental integration strategies are discussed.

11.6.1 Top down Integration

Top down integration testing is an incremental approach to construction or program structure. Modules are integrated by moving downward through the control hierarchy beginning with the main control module (main program). Modules subordinate (and ultimately subordinate) to the main control module are incorporated into the structure in either a depth-first or breadth-first manner.
Referring to Figure below depth-first integration would integrate all components on a major control path of the structure. Selection of a major path is somewhat arbitrary and depends on application-

specific characteristics. For example, selecting the left hand path, components M1 M2 and M5 would be integrated first. Next, M8 or (if necessary proper functioning of M2) M6 would be integrated. Then, the central and right hand control paths are built. Breadth first integration incorporates all components directly sub-ordinate at each level, moving across the structure horizontally.



The integration process is performed in a series of five steps:

1. The main control module is used as a test driver and stubs are substituted for all components directly sub-ordinate to the main control module.
2. Depending on the integration approach selected (i.e., depth or breadth first), sub-ordinate stubs are replaced one at a time with actual components.
3. Tests are conducted as each component is integrated.
4. On completion of each set of tests, another stub is replaced with the real components.
5. Regression testing may be conducted to ensure that new errors have not been introduced.

The process continues from step 2 until the entire program structure is built.

The top-down integration strategy verifies major control or decision points early in the test process. In a well-factored program structure, decision making occurs at upper levels in the hierarchy and is therefore encountered first. If major control problems do exist, early recognition is essential. If depth-first integration is selected, a complete function of the software may be implemented and demonstrated.

Top-down strategy sounds relatively uncomplicated, but in practice, logistical problems can arise. The most common of these problems occurs when processing at low levels in the hierarchy is required to adequately test upper levels. Stubs replace low-level modules at the beginning of top-down testing; therefore, no significant data can now upward in the program structure. The tester is left will three choices;

- Delay many tests until stubs are replaced with actual modules.
- Develop stubs that perform limited functions that simulate the actual module, or
- Integrate the software from the bottom of the hierarchy upward.

The first approach (delay tests until stubs are replaced by actual modules) causes us to loose some control over correspondence between specific tests and incorporation of specific modules. This can lead to difficulty in determining the cause of errors and tends to violate the highly constrained nature of the top-down approach. The second approach is workable but can lead to significant overhead, as stubs become more and more complex. The third approach is called bottom-up testing.

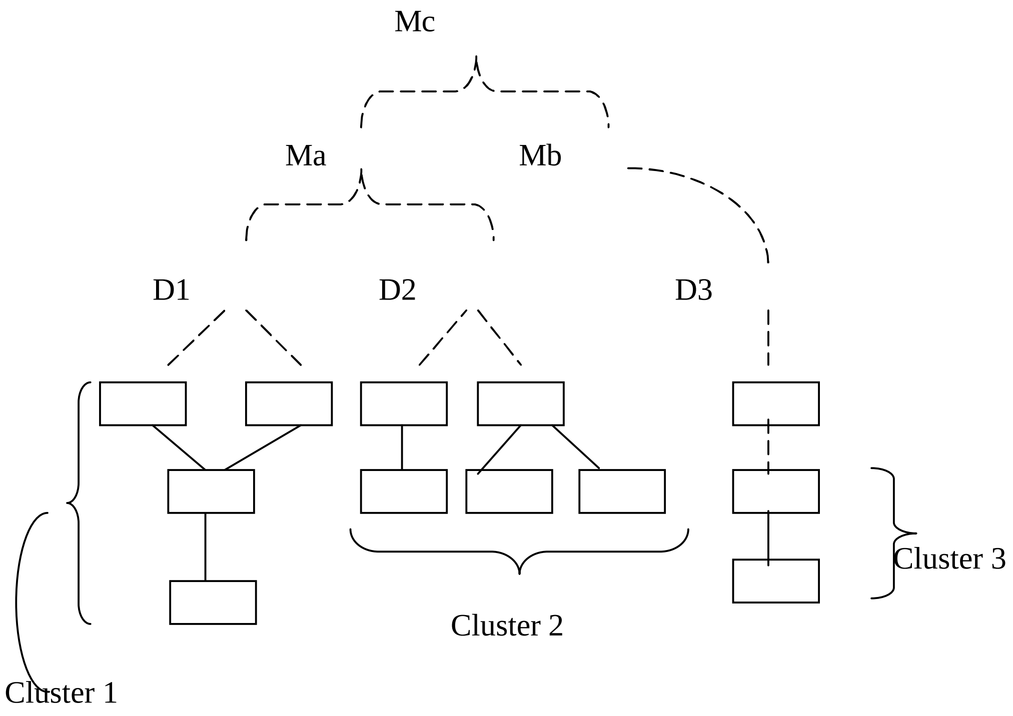
11.6.2 Bottom-up Integration

Bottom-up integration testing, as its name implies, begins construction and testing with atomic *modules* (i.e., components at the lowest levels in the program structure). Because components are integrated from the bottom up, processing required for components subordinate to a given level is always available and the need for stubs is eliminated.

A bottom-up integration strategy may be implemented with the following steps:

- Low-level components are combined into clusters (sometimes called *builds) that perform a specific software sub-function.*
- A driver (a control program for testing) is written to coordinate test case input and output.
- The cluster is tested.
- Drivers are removed and clusters are combined moving upward in the program structure.

Integration follows the pattern illustrated in Figure below. Components are combined to form clusters 1,2, and 3.



Each of the clusters is tested using a driver (shown as a dashed block). Components in clusters 1 and 2 are subordinate to Ma. Drivers D1 and D2 are removed and the clusters are interfaced directly to Ma. Similarly, driver D3 for cluster 3 is removed prior to integration with module Mb. Both Ma and Mb will ultimately be integrated with component Mc, and so forth.

As integration moves upward the need for separate test drivers lessens. In fact, if the top two levels of program structure are integrated top down, the number of drivers can be reduced substantially and integration of clusters is greatly simplified.

11.7 REGRESSION TESTING

Each time a new module is added as part of integration testing, the software changes. New data flow paths are established, new I/O may occur, and new control logic is invoked. These changes may cause problems with functions that previously worked flawlessly. In the context of an integration test strategy, *regression testing* is the re-execution of some subset of tests that have already been conducted to ensure that changes have not propagated unintended side effects.

In a broader context, successful tests (of any kind) result in the discovery of errors, and errors must be corrected. Whenever software is corrected, some aspect of the software configuration (the program, its documentation, or the data that support it) is changed. Regression testing is the activity that hclp5 to en5ure that changes (due to testing or for other reasons) do not introduce unintended behaviour or additional errors.

For instance, suppose you are going to add new functionality to your software, or you are going to modify a module to improve its response time. The changes, of course, may introduce errors into software that was previously correct. For example, suppose the program fragment

```
x := c + 1 ;
proc (z);
c := x + 2; x:= 3;
```

works properly. Now suppose that in a subsequent redesign it is transformed into

```
proc(z);
c := c + 3;
x:= 3;
```

in an attempt at program optimization. This may result in an error if procedure proc accesses variable x.

Thus, we need to organize testing also with the purpose of verifying possible *regressions* of software during its life, i.e., degradations of correctness or other qualities due to later modifications. Properly designing and documenting test cases with the purpose of making tests repeatable, and using test generators, will help regression testing. Conversely, the use of interactive human input reduces repeatability and thus hampers regression testing.

Finally, we must treat test cases in much the same way as software. It is clear that such factors as resolvability, reusability, and verifiability are just as important in test cases as they are in software. We must apply formality and rigor and all of our other principles in the development and management of test cases.

11.8 COMMENTS ON INTEGRATION TESTING

There has been much discussion of the relative advantages and disadvantages of top-down versus bottom-up integration testing. In general, the advantages of one strategy tend to result in disadvantages for the other strategy. The major disadvantage of the top-down approach is the need for stubs and the attendant testing difficulties that can be associated with them. Problems associated with stubs may be offset by the advantage of testing major control functions early. The major disadvantage of bottom-up integration is that the program as an entity does not exist until the last module is added. This drawback is tempered by easier test case design and a lack of stubs.

Selection of an integration strategy depends upon software characteristics and, sometimes, project schedule. In general, a combined approach (sometimes called *sandwich testing) that* uses top-down tests for upper levels of the program structure, coupled with bottom-up tests for subordinate levels may be the best compromise.

As integration testing is conducted, the tester should identify *critical modules.* A critical module has one or more of the following characteristics:

- addresses several software requirements,
- has a high level of control (resides relatively high in the program structure),
- is complex or error prone (cyclomatic complexity may be used as an indicator), or
- has definite performance requirements.
- Critical modules should be tested as early as is possible.

In addition, regression tests should focus on critical module function.

11.9 THE ART OF DEBUGGING

Software testing is a process that can be systematically planned and specified. Test case design can be conducted, a strategy can be defined, and results can be evaluated against prescribed expectations.

Debugging occurs as a consequence or successful testing. That is, when a test case uncovers an error, debugging is the process that results in the removal or the error. Although debugging can and should be an orderly process, it is still very much an art. A software engineer, evaluating the results or a test, is often confronted with a "symptomatic" indication or a software problem. That is, the external manifestation or the error and the internal cause or the error may have no obvious relationship to one another. The poorly understood mental process that connects a symptom to a cause is debugging.

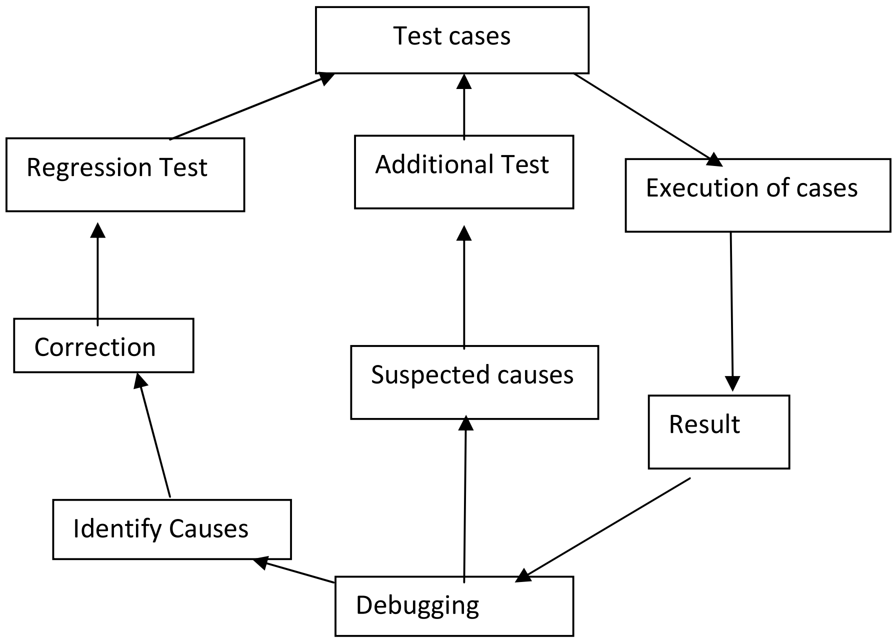
11.9.1 The Debugging Process

Debugging is not testing but always occurs as a consequence of testing. Referring to Figure in the next page, the debugging process begins with the execution or a test case. Results are assessed and a lack or correspondence between expected and actual performance is encountered. In many cases, the non-corresponding data are a symptom of an underlying cause as yet hidden. The debugging process attempts to match symptom with cause thereby leading to error correction.

The debugging process will always have one or two outcomes:

1. The cause will be found and corrected, *or*
2. The cause will not be found.

In the latter case, the person performing debugging may suspect a cause, design a test case to help validate that suspicion, and work toward error correction in an iterative fashion.



Why is debugging so difficult? In all likelihood, human psychology has more to do with an answer than software technology. However, a few characteristics or bugs provide some clues:

- The symptom and the cause may be geographically remote. That is, the symptom may appear in one part or a program, while the cause may actually be located at a site that is far removed. Highly coupled program structures exacerbate this situation.

- The symptom may disappear (temporarily) when another error is corrected.
- The symptom may actually be caused by non-errors (e.g., round-off inaccuracies).
- The symptom may be caused by human error that is not easily traced.
- The symptom may be a result, or timing problems, rather than processing problems.
- It may be difficult to accurately reproduce input conditions (e.g., a real-time application in which input ordering is indeterminate).
- The symptom may be intermittent. This is particularly common in embedded systems that couple hardware and software inextricably.
- The symptom may be due to causes that are distributed across a number of tasks running on different processors.

During debugging, we encounter errors that range from mildly annoying (e,g., an incorrect output format) to catastrophic (e.g. the system fails, causing serious economic or physical damage). As the consequences or an error increase, the amount of pressure to find the cause also increases. Often, pressure sometimes forces a sort- ware developer to fix one error and at the same time introduce two more.

10.10 SUMMARY:

In this way, we learned above Strategic approach to software testing, Unit Testing, Integration Testing and Debugging process in detail.

Questions:

1. Explain Unit Testing?

Ans: Refer 11.5

2. Explain Integration testing?

Ans: Refer 11.6

3. Explain Regression Testing?

Ans: Refer 11.7

4. Explain Debugging Process in detail?

Ans: Refer 11.9.1

❖❖❖❖

12

CATEGORIES OF TESTING

Unit Structure

12.1 INTRODUCTION:

In this Chapter, we will learn the testing life cycle of software and also testing methods like white box testing and black box testing. Also we trying to cover the sub processes of white box testing and black box testing methods such as Integration Testing, Unit Testing, Regression Testing, System Testing and much more.

12.2 THE TESTING PROCESS AND THE SOFTWARE TESTING LIFE CYCLE:

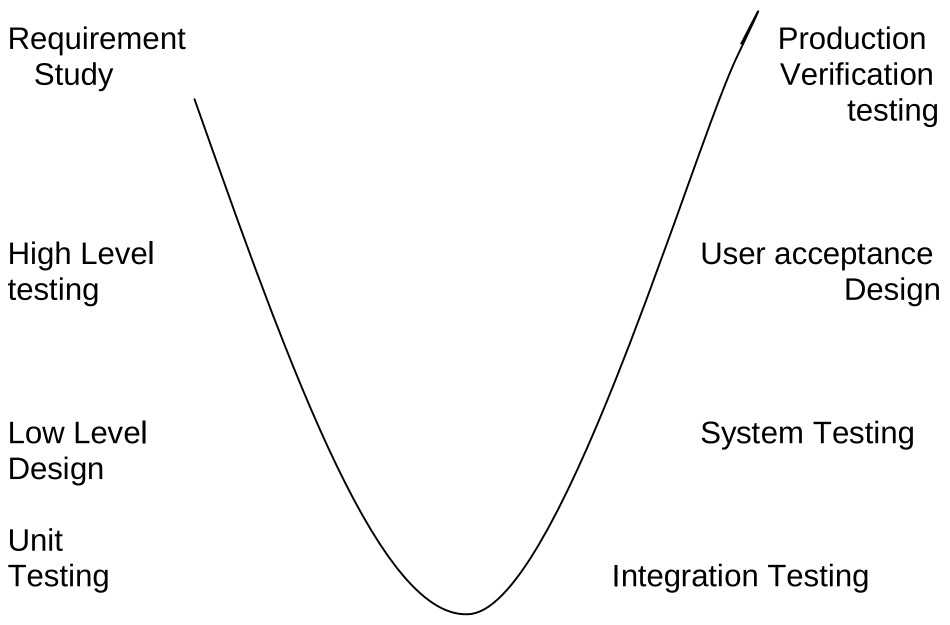
Every testing project has to follow the waterfall model of the testing process.
The waterfall model is as given below
1. Test Strategy & Planning

2. Test Design
3. Test Environment setup
4. Test Execution
5. Defect Analysis & Tracking
6. Final Reporting

According to the respective projects, the scope of testing can be tailored, but the process mentioned above is common to any testing activity.

Software Testing has been accepted as a separate discipline to the extent that there is a separate life cycle for the testing activity.

Involving software testing in all phases of the software development life cycle has become a necessity as part of the software quality assurance process. Right from the Requirements study till the implementation, there needs to be testing done on every phase. The V-Model of the Software Testing Life Cycle along with the Software Development Life cycle given below indicates the various phases or levels of testing.



SDLC - STLC

There are two categories of testing activities that can be done on software, namely,

- **Static Testing**
- **Dynamic Testing**

The kind of verification we do on the software work products before the process of Compilation and creation of an executable is more of Requirement review, design review, code review, walkthrough and audits. This type of testing is called *Static Testing*. When we test the software by executing and comparing the actual & expected results, it is called *Dynamic Testing*

12.3 TYPES OF TESTING

From the V-model, we see that are various levels or phases of testing, namely, Unit testing, Integration testing, System testing, User Acceptance testing etc.

Let us see a brief definition on the widely employed types of testing.

Unit Testing: The testing done to a unit or to a smallest piece of software. Done to verify if it satisfies its functional specification or its intended design structure.

Integration Testing: Testing which takes place as sub elements are combined (i.e. integrated) to form higher-level elements

Regression Testing: Selective re-testing of a system to verify the modification (bug fixes) have not caused unintended effects and that system still complies with its specified requirements

System Testing: Testing the software for the required specifications on the intended hardware

Acceptance Testing: Formal testing conducted to determine whether or not a system satisfies its acceptance criteria, which enables a customer to determine whether to accept the system or not.

Performance Testing: To evaluate the time taken or response time of the system to perform it’s required functions in comparison

Stress Testing: To evaluate a system beyond the limits of the specified requirements or ystem resources (such as disk space, memory, processor utilization) to ensure the system do not reak unexpectedly

Load Testing: Load Testing, a subset of stress testing, verifies that a web site can handle a particular number of concurrent users while maintaining acceptable response times

Alpha Testing: Testing of a software product or system conducted at the developer’s site by the customer

Beta Testing: Testing conducted at one or more customer sites by the end user of a delivered software product system.

12.4 THE TESTING TECHNIQUES

To perform these types of testing, there are two widely used testing techniques. The above said testing types are performed based on the following testing techniques.

Black-Box testing technique:

This technique is used for testing based solely on analysis of requirements (specification, user documentation.). Also known as functional testing.

White-Box testing technique:

This technique us used for testing based on analysis of internal logic (design, code, etc.)(But expected results still come requirements). Also known as structural testing.

12.5 BLACK BOX AND WHITE BOX TESTING:

Test Design refers to understanding the sources of test cases, test coverage, how to develop and document test cases, and how to build and maintain test data. There are 2 primary methods by which tests can be designed and they are:

- BLACK BOX
- WHITE BOX

Black-box test design treats the system as a literal "black-box", so it doesn't explicitly use knowledge of the internal structure. It is usually described as focusing on testing functional requirements. Synonyms for black-box include: behavioural, functional, opaque -box, and closed-box.

White-box test design allows one to peek inside the "box", and it focuses specifically on using internal knowledge of the software to guide the selection of test data. It is used to detect errors by means of execution-oriented test cases.

Synonyms for white-box include:

Structural, glass-box and clear-box.

While black-box and white-box are terms that are still in popular use, many people prefer the terms "behavioural" and "structural". Behavioural test design is slightly different from black-box test design because the use of internal knowledge isn't strictly forbidden, but it's still discouraged. In practice, it hasn't proven useful to use a single test design method. One has to use a mixture of different methods so that they aren't hindered by the limitations of a particular one. Some call this "gray-box" or "translucent-box" test design, but others wish we'd stop talking about boxes altogether!!!

12.6 BLACK BOX TESTING

Black Box Testing is testing without knowledge of the internal workings of the item being tested. For example, when black box testing is applied to software engineering, the tester would only know the "legal" inputs and what the expected outputs should be, but not how the program actually arrives at those outputs. It is because of this that black box testing can be considered testing with respect to the specifications, no other knowledge of the program is necessary. For this reason, the tester and the programmer can be independent of one another, avoiding programmer bias toward his own work. For this testing, test groups are often used, Though centered around the knowledge of user requirements, black box tests do not necessarily involve the participation of users. Among the most important black box tests that do not involve users are functionality testing, volume tests, stress tests, recovery testing, and benchmarks. Additionally, there are two types of black box test that involve users, i.e. field and laboratory tests. In the following the most important aspects of these black box tests will be described briefly.

- **Black box testing - without user involvement**

The so-called ``functionality testing'' is central to most testing exercises. Its primary objective is to assess whether the program does what it is supposed to do, i.e. what is specified in the requirements. There are different approaches to functionality testing. One is the testing of each program feature or function in sequence. The other is to test module by module, i.e. each function where it is called first.

The objective of volume tests is to find the limitations of the software by processing a huge amount of data. A volume test can uncover problems that are related to the *efficiency* of a system, e.g. incorrect buffer sizes, a consumption of too much memory space, or only show that an error message would be needed telling the user that the system cannot process the given amount of data.

During a stress test, the system has to process a huge amount of data or perform many function calls within a short period of time. A typical example could be to perform the same function from all workstations connected in a LAN within a short period of time (e.g. sending e-mails, or, in the NLP area, to modify a term bank via different terminals simultaneously).

The aim of recovery testing is to make sure to which extent data can be recovered after a system breakdown. Does the system provide possibilities to recover all of the data or part of it? How much can be recovered and how? Is the recovered data still correct and consistent? Particularly for software that needs high *reliability* standards, recovery testing is very important.

The notion of benchmark tests involves the testing of program *efficiency*. The efficiency of a piece of software strongly depends on the hardware environment and therefore benchmark tests always consider the soft/hardware combination. Whereas for most software engineers benchmark tests are concerned with the quantitative measurement of specific operations, some also consider user tests that compare the efficiency of different software systems as benchmark tests. In the context of this document, however, benchmark tests only denote operations that are independent of personal variables.

- **Black box testing - with user involvement**

For tests involving users, methodological considerations are rare in SE literature. Rather, one may find practical test reports that distinguish roughly between field and laboratory tests. In the following only a rough description of field and laboratory tests will be given.

E.g. Scenario Tests. The term ``scenario'' has entered software evaluation in the early 1990s . A scenario test is a test case which aims at a realistic user background for the evaluation of software as it was defined and performed It is an instance of black box testing where the major objective is to assess the suitability of a software product for every-day routines. In short it involves putting the system into its intended use by its envisaged type of user, performing a standardised task.

In field tests users are observed while using the software system at their normal working place. Apart from general *usability*-related aspects, field tests are particularly useful for assessing the *interoperability* of the software system, i.e. how the technical integration of the system works. Moreover, field tests are the only real means to elucidate problems of the organisational integration of the software system into existing procedures. Particularly in the NLP environment this problem has frequently been underestimated. A typical example of the organisational problem of implementing a translation memory is the language service of a big automobile manufacturer, where the major implementation problem is not the technical environment, but the fact that many clients still submit their orders as print-out, that neither source texts nor target texts are properly organised and stored and, last but not least, individual translators are not too motivated to change their working habits.

Laboratory tests are mostly performed to assess the general *usability* of the system. Due to the high laboratory equipment costs laboratory tests are mostly only performed at big software houses such as IBM or Microsoft. Since laboratory tests provide testers with many technical possibilities, data collection and analysis are easier than for field tests.

12.6.1 Black box testing Methods

- **Graph-based Testing Methods**

✓ Black-box methods based on the nature of the relationships (links) among the program objects (nodes), test cases are designed to traverse the entire graph

✓ Transaction flow testing (nodes represent steps in some transaction and links represent logical connections between steps that need to be validated)

✓ Finite state modelling (nodes represent user observable states of the software and links represent transitions between states)

✓ Data flow modelling (nodes are data objects and links are transformations from one data object to another)

✓ Timing modelling (nodes are program objects and links are sequential connections between these objects, link weights are required execution times)

- **Equivalence Partitioning**

Black-box technique that divides the input domain into classes of data from which test cases can be derived. An ideal test case uncovers a class of errors that might require many arbitrary test cases to be executed before a general error is observed

Equivalence class guidelines:

1. If input condition specifies a range, one valid and two invalid equivalence classes are defined

2. If an input condition requires a specific value, one valid and two invalid equivalence classes are defined

3. If an input condition specifies a member of a set, one valid and one invalid equivalence class is defined

4. If an input condition is Boolean, one valid and one invalid equivalence class is defined

- **Comparison Testing**

Black-box testing for safety critical systems in which independently developed implementations of redundant systems are tested for conformance to specifications. Often equivalence class partitioning is used to develop a common set of test cases for each implementation.

- **Orthogonal Array Testing**

Black-box technique that enables the design of a reasonably small set of test cases that provide maximum test coverage. Focus is on categories of faulty logic likely to be present in the software component (without examining the code) · Priorities for assessing tests using an orthogonal array
1. Detect and isolate all single mode faults
2. Detect all double mode faults
3. Multimode faults

12.6.2 Advantages of Black Box Testing

· More effective on larger units of code than glass box testing

· Tester needs no knowledge of implementation, including specific programming languages

· Tester and programmer are independent of each other

· Tests are done from a user's point of view

· Will help to expose any ambiguities or inconsistencies in the specifications

· Test cases can be designed as soon as the specifications are complete

12.6.3 Disadvantages of Black Box Testing

· Only a small number of possible inputs can actually be tested, to test every possible input stream would take nearly forever

· Without clear and concise specifications, test cases are hard to design

· There may be unnecessary repetition of test inputs if the tester is not informed of test cases the programmer has already tried

· May leave many program paths untested

· Cannot be directed toward specific segments of code which may be very complex (and therefore more error prone)

· Most testing related research has been directed toward glass box testing

12.7 WHITE BOX TESTING

Software testing approaches that examine the program structure and derive test data from the program logic. Structural testing is sometimes referred to as clear-box testing since white boxes are considered opaque and do not really permit visibility into the code.

- **Synonyms for white box testing**

· Glass Box testing
· Structural testing
· Clear Box testing
· Open Box Testing

- **The purpose of white box testing**

Initiate a strategic initiative to build quality throughout the life cycle of a software product or service.

Provide a complementary function to black box testing.

Perform complete coverage at the component level.

Improve quality by optimizing performance.

12.7.1 Code Coverage Analysis

- **Basis Path Testing**

A testing mechanism proposed by McCabe whose aim is to derive a logical complexity measure of a procedural design and use this as a guide for defining a basic set of execution paths. These are test cases that exercise basic set will execute every statement at least once.

- **Flow Graph Notation**

A notation for representing control flow similar to flow charts and UML activity diagrams.

- **Cyclomatic Complexity**

The cyclomatic complexity gives a quantitative measure of 4the logical complexity. This value gives the number of independent paths in the basis set, and an upper bound for the number of tests to ensure that each statement is executed at least once. An independent path is any path through a program that introduces at least one new set of processing statements or a new condition (i.e., a new edge). Cyclomatic complexity provides upper bound for number of tests required to guarantee coverage of all program statements.

12.7.2 Control Structure testing

- **Conditions Testing**

Condition testing aims to exercise all logical conditions in a program module. They may define:

- ✓ Relational expression: (E1 op E2), where E1 and E2 are arithmetic expressions.
- ✓ Simple condition: Boolean variable or relational expression, possibly proceeded by a NOT operator.
- ✓ Compound condition: composed of two or more simple conditions, Boolean operators and parentheses.
- ✓ Boolean expression: Condition without Relational expressions.

- **Data Flow Testing**

Selects test paths according to the location of definitions and use of variables.

- **Loop Testing**

Loops fundamental to many algorithms. Can define loops as imple, concatenated, nested, and unstructured.

Examples:
Note that unstructured loops are not to be tested . rather, they are redesigned.

- **Design by Contract (D b C)**

DbC is a formal way of using comments to incorporate specification information into the code itself. Basically, the code specification is expressed unambiguously using a formal language that describes the code's implicit contracts. These contracts specify such requirements as:

- ✓ Conditions that the client must meet before a method is invoked.
- ✓ Conditions that a method must meet after it executes.
- ✓ Assertions that a method must satisfy at specific points of its execution

- **Profiling**

Profiling provides a framework for analyzing Java code performance for speed and heap memory use. It identifies routines that are consuming the majority of the CPU time so that problems may be tracked down to improve performance.

- **Error Handling**

Exception and error handling is checked thoroughly are simulating partial and complete fail-over by operating on error causing test vectors. Proper error recovery, notification and logging are checked against references to validate program design.

- **Transactions**

Systems that employ transaction, local or distributed, may be validated to ensure that ACID (Atomicity, Consistency, Isolation, Durability). Each of the individual parameters is tested individually against a reference data set.

Transactions are checked thoroughly for partial/complete commits and rollbacks encompassing databases and other XA compliant transaction processors.

12.7.3 Advantages of White Box Testing

· Forces test developer to reason carefully about implementation

· Approximate the partitioning done by execution equivalence

· Reveals errors in "hidden" code

· Beneficent side-effects

12.7.4 Disadvantages of White Box Testing

· Expensive

· Cases omitted in the code could be missed out.

12.8 DIFFERENCE BETWEEN BLACK BOX TESTING AND WHITE BOX TESTING

An easy way to start up a debate in a software testing forum is to ask the difference between black box and white box testing. These terms are commonly used, yet everyone seems to have a different idea of what they mean.

Black box testing begins with a metaphor. Imagine you're testing an electronics system. It's housed in a black box with lights, switches, and dials on the outside. You must test it without opening it up, and you can't see beyond its surface. You have to see if it works just by flipping switches (inputs) and seeing what happens to the lights and dials (outputs). This is black box testing. Black box software testing is doing the same thing, but with software. The actual meaning of the metaphor, however, depends on how you define the boundary of the box and what kind of access the "blackness" is blocking.

An opposite test approach would be to open up the electronics system, see how the circuits are wired, apply probes internally and maybe even disassemble parts of it. By analogy, this is called white box testing, To help understand the different ways that software testing can be divided between black box and white box techniques, consider the Five-Fold Testing System. It lays out five dimensions that can be used for examining testing:

1. People (who does the testing)
2. Coverage (what gets tested)
3. Risks (why you are testing)
4. Activities (how you are testing)
5. Evaluation (how you know you've found a bug)

Let's use this system to understand and clarify the characteristics of black box and white box testing.

People: Who does the testing?
Some people know how software works (developers) and others just use it (users).

Accordingly, any testing by users or other non-developers is sometimes called "black box" testing. Developer testing is called "white box" testing. The distinction here is based on what the person knows or can understand.

Coverage: What is tested?
If we draw the box around the system as a whole, "black box" testing becomes another name for system testing. And testing the units inside the box becomes white box testing.

This is one way to think about coverage. Another is to contrast testing that aims to cover all the requirements with testing that aims to cover all the code. These are the two most commonly used coverage criteria. Both are supported by extensive literature and commercial tools. Requirements-based testing could be called "black box" because it makes sure that all the customer requirements have been verified. Code-based testing is often called "white box" because it makes sure that all the code (the statements, paths, or decisions) is exercised.

Risks: Why are you testing?
Sometimes testing is targeted at particular risks. Boundary testing and other attack-based techniques are targeted at common coding errors. Effective security testing also requires a detailed understanding of the code and the system architecture. Thus, these techniques might be classified as "white box". Another set of risks concerns whether the software will actually provide value to users. Usability testing focuses on this risk, and could be termed "black box."

Activities: How do you test?

A common distinction is made between behavioural test design, which defines tests based on functional requirements, and structural test design, which defines tests based on the code itself. These are two design approaches. Since behavioural testing is based on external functional definition, it is often called “black box,” while structural testing—based on the code internals—is called “white box.” Indeed, this is probably the most commonly cited definition for black box and white box testing. Another activity-based distinction contrasts dynamic test execution with formal code inspection. In this case, the metaphor maps test execution (dynamic testing) with black box testing, and maps code inspection (static testing) with white box testing. We could also focus on the tools used. Some tool vendors refer to code-coverage tools as white box tools, and tools that facilitate applying inputs and capturing inputs—most notably GUI capture replay tools—as black box tools. Testing is then categorized based on the types of tools used.

Evaluation: How do you know if you’ve found a bug?

There are certain kinds of software faults that don’t always lead to obvious failures. They may be masked by fault tolerance or simply luck. Memory leaks and wild pointers are examples. Certain test techniques seek to make these kinds of problems more visible. Related techniques capture code history and stack information when faults occur, helping with diagnosis. Assertions are another technique for helping to make problems more visible. All of these techniques could be considered white box test techniques, since they use code instrumentation to make the internal workings of the software more visible.

These contrast with black box techniques that simply look at the official outputs of a program.

White box testing is concerned only with testing the software product, it cannot guarantee that the complete specification has been implemented. Black box testing is concerned only with testing the specification, it cannot guarantee that all parts of the implementation have been tested. Thus black box testing is testing against the specification and will discover faults of omission, indicating that part of the specification has not been fulfilled.

White box testing is testing against the implementation and will discover faults of commission, indicating that part of the implementation is faulty. In order to fully test a software product both black and white box testing are required.

White box testing is much more expensive than black box testing. It requires the source code to be produced before the tests can be planned and is much more laborious in the determination of suitable input data and the determination if the software is or is not correct. The advice given is to start test planning with a black box test approach as soon as the specification is available. White box planning should commence as soon as all black box tests have been successfully passed, with the production of flow graphs and determination of paths. The paths should then be checked against the black box test plan and any additional required test runs determined and applied.

The consequences of test failure at this stage may be very expensive. A failure of a white box test may result in a change which requires all black box testing to be repeated and the re-determination of the white box paths

12.9 SUMMARY:

To conclude, apart from the above described analytical methods of both white and black box testing, there are further constructive means to guarantee high quality software end products. Among the most important constructive means are the usages of object-oriented programming tools, the integration of CASE tools, rapid prototyping, and last but not least the involvement of users in both software development and testing procedures.

Questions:

1. Explain Software Testing Life Cycle in detail?
Ans: Refer 12.2

2. Explain Types of Testing in detail?
Ans: Refer 12.3

3. Explain Black Box and White Box testing?
Ans: Refer 12.5

❖❖❖❖

13

SOFTWARE TESTING

Unit Structure

13.1 INTRODUCTION:

Software testing is an art. Most of the testing methods and practices are not very different from 20 years ago. It is nowhere near maturity, although there are many tools and techniques available to use. Good testing also requires a tester's creativity, experience and intuition, together with proper techniques

Before moving further towards introduction to software testing, we need to know a few concepts that will simplify the definition of software testing.

- **Error**: Error or mistake is a human action that produces wrong or incorrect result.
- **Defect (Bug, Fault)**: A flaw in the system or a product that can cause the component to fail.
- **Failure**: It is the variance between the actual and expected result.
- **Risk**: Risk is a factor that could result in negativity or a chance of loss or damage.

Thus Software testing is the process of finding defects/bugs in the system, which occurs due to an error in the application, which could lead to failure of the resultant product and increase in probability of high risk. In short, software testing has different goals and objectives, which often include:

1. finding defects;
2. gaining confidence in and providing information about the level of quality;
3. Preventing defects.

13.2 SCOPE OF SOFTWARE TESTING

The primary function of software testing is to detect bugs in order to correct and uncover it. The scope of software testing includes execution of that code in various environments and also to examine the aspects of code - does the software do what it is supposed to do and function according to the specifications? As we move further we come across some questions such as "When to start testing?" and "When to stop testing?" It is recommended to start testing from the initial stages of the software development. This not only helps in rectifying tremendous errors before the last stage, but also reduces the rework of finding the bugs in the initial stages every now and then. It also saves the cost of the defect required to find it. Software testing is an ongoing process, which is potentially endless but has to be stopped somewhere, due to the lack of time and budget. It is required to achieve maximum profit with good quality product, within the limitations of time and money. The tester has to follow some procedural way through which he can judge if he covered all the points required for testing or missed out any.

13.3 SOFTWARE TESTING KEY CONCEPTS

- **Defects and Failures**: As we discussed earlier, defects are not caused only due to the coding errors, but most commonly due to the requirement gaps in the non-functional requirement, such as usability, testability, scalability, maintainability, performance and security. A failure is caused due to the deviation between an actual and an expected result. But not all defects result to failures. A defect can turn into a failure due to the change in the environment and or the change in the configuration of the system requirements.

- **Input Combination and Preconditions**: Testing all combination of inputs and initial state (preconditions), is not feasible. This means finding large number of infrequent defects is difficult.

- **Static and Dynamic Analysis**: Static testing does not require execution of the code for finding defects, whereas in dynamic testing, software code is executed to demonstrate the results of running tests.

- **Verification and Validation**: Software testing is done considering these two factors.

  1. Verification: This verifies whether the product is done according to the specification?

  2. Validation: This checks whether the product meets the customer requirement?

- **Software Quality Assurance**: Software testing is an important part of the software quality assurance. Quality assurance is an activity, which proves the suitability of the product by taking care of the quality of a product and ensuring that the customer requirements are met.

13.4 SOFTWARE TESTING TYPES

Software test type is a group of test activities that are aimed at testing a component or system focused on a specific test objective; a non-functional requirement such as usability, testability or reliability. Various types of software testing are used with the common objective of finding defects in that particular component.

Software testing is classified according to two basic types of software testing: Manual Scripted Testing and Automated Testing.

Manual Scripted Testing:

- Black Box Testing
- White Box Testing
- Gray Box Testing

The level of software testing life cycle includes:

- Unit Testing
- Integration Testing
- System Testing
- Acceptance Testing
  1. Alpha Testing
  2. Beta Testing

Other types of software testing are:

- Functional Testing
- Performance Testing

    1. Load Testing
    2. Stress Testing
- Smoke Testing
- Sanity Testing
- Regression Testing
- Recovery Testing
- Usability Testing
- Compatibility Testing
- Configuration Testing
- Exploratory Testing

For further explanation of these concepts, read more on types of software testing.

Automated Testing: Manual testing is a time consuming process. Automation testing involves automating a manual process. Test automation is a process of writing a computer program in the form of scripts to do a testing which would otherwise need to be done manually. Some of the popular automation tools are Winrunner, Quick Test Professional (QTP), Load Runner, Silk Test, Rational Robot, etc. Automation tools category also includes maintenance tool such as Test Director and many other.

13.5 SOFTWARE TESTING METHODOLOGIES

The software testing methodologies or process includes various models that built the process of working for a particular product. These models are as follows:

- Waterfall Model
- V Model
- Spiral Model
- Rational Unified Process(RUP)
- Agile Model
- Rapid Application Development(RAD)

These models are elaborated briefly in software testing methodologies.

13.6 SOFTWARE TESTING ARTIFACTS

Software testing process can produce various artifacts such as:

- **Test Plan**: A test specification is called a test plan. A test plan is documented so that it can be used to verify and ensure that a product or system meets its design specification.

- **Traceability matrix**: This is a table that correlates or design documents to test documents. This verifies that the test results are correct and is also used to change tests when the source documents are changed.
- **Test Case**: Test cases and software testing strategies are used to check the functionality of individual component that is integrated to give the resultant product. These test cases are developed with the objective of judging the application for its capabilities or features.
- **Test Data**: When multiple sets of values or data are used to test the same functionality of a particular feature in the test case, the test values and changeable environmental components are collected in separate files and stored as test data.
- **Test Scripts**: The test script is the combination of a test case, test procedure and test data.
- **Test Suite**: Test suite is a collection of test cases.

Software Testing Process

Software testing process is carried out in the following sequence, in order to find faults in the software system:

1. Create Test Plan
2. Design Test Case
3. Write Test Case
4. Review Test Case
5. Execute Test Case
6. Examine Test Results
7. Perform Post-mortem Reviews
8. Budget after Experience

13.7 AVAILABLE TOOLS, TECHNIQUES, AND METRICS

- There are an abundance of software testing tools exist. The correctness testing tools are often specialized to certain systems and have limited ability and generality. Robustness and stress testing tools are more likely to be made generic.

- *Mothora* [DeMillo91] is an automated mutation testing tool-set developed at Purdue University. Using Mothora, the tester can create and execute test cases, measure test case adequacy, determine input-output correctness, locate and remove faults or bugs, and control and document the test.

- *NuMega's Boundschecker* [NuMega99] *Rational's Purify* [Rational99]. They are run-time checking and debugging aids. They can both check and protect against memory leaks and pointer problems.

- *Ballista COTS Software Robustness Testing Harness* [Ballista99]. The Ballista testing harness is a full-scale automated robustness testing tool. The first version supports testing up to 233 POSIX function calls in UNIX operating systems. The second version also supports testing of user functions provided that the data types are recognized by the testing server. The Ballista testing harness gives quantitative measures of robustness comparisons across operating systems. The goal is to automatically test and harden Commercial Off-The-Shelf (COTS) software against robustness failures.

13.8 SUMMARY:

Software testing is an art. Most of the testing methods and practices are not very different from 20 years ago. It is nowhere near maturity, although there are many tools and techniques available to use. Good testing also requires a tester's creativity, experience and intuition, together with proper techniques.

Questions:
1. Explain Software Testing Key Concepts?
Ans: refer

2. Explain Software Testing Methodologies
Ans: refer 13.5

3. Explain Available tools, techniques in detail?
Ans: refer 13.7

❖❖❖❖

14

IMPLEMENTATION & MAINTENANCE

Unit Structure

14.1 INTRODUCTION:

The quality of data input agrees on the quality of information output. Systems analysts can support accurate data entry through success of three broad objectives: effective coding, effective and efficient data capture and entry, and assuring quality through validation. In this Chapter, we will learn Data Entry and Data Format.

14.2 DATA ENTRY AND DATA STORAGE

The quality of data input determines the quality of information output. Systems analysts can support accurate data entry through achievement of three broad objectives: effective coding, effective and efficient data capture and entry, and assuring quality through validation. Coding aids in reaching the objective of efficiency, since data that are coded require less time to enter and reduce the number of items entered. Coding can also help in appropriate sorting of data during the data transformation process. Additionally, coded data can save valuable memory and storage space.

In establishing a coding system, systems analysts should follow these guidelines:
•Keep codes concise.
•Keep codes stable.

•Make codes that are unique.
•Allow codes to be sort.
•Avoid confusing codes.
•Keep codes uniform.
•Allow for modification of codes.
•Make codes meaningful.

The simple sequence code is a number that is assigned to something if it needs to be numbered. It therefore has no relation to the data itself. Classification codes are used to distinguish one group of data, with special characteristics, from another. Classification codes can consist of either a single letter or number. The block sequence code is an extension of the sequence code. The advantage of the block sequence code is that the data are grouped according to common characteristics, while still taking advantage of the simplicity of assigning the next available number within the block to the next item needing identification.

A mnemonic is a memory aid. Any code that helps the data-entry person remembers how to enter the data or the end-user remembers how to use the information can be considered a mnemonic. Mnemonic coding can be less arbitrary, and therefore easier to remember, than numeric coding schemes. Compare, for example, a gender coding system that uses "F" for Female and "M" for Male with an arbitrary numeric coding of gender where perhaps "1" means Female and "2" means Male. Or, perhaps it should be "1" for Male and "2" for Female? Or, why not "7" for Male and "4" for Female? The arbitrary nature of numeric coding makes it more difficult for the user.

14.3 DATE FORMATS

An effective format for the storage of date values is the eight-digit YYYYMMDD format as it allows for easy sorting by date. Note the importance of using four digits for the year. This eliminates any ambiguity in whether a value such as 01 means the year 1901 or the year 2001. Using four digits also insures that the correct sort sequence will be maintained in a group of records that include year values both before and after the turn of the century (e.g., 1999, 2000, 2001).

Remember, however, that the date format you use for storage of a date value need not be the same date format that you present to the user via the user interface or require of the user for data entry. While YYYYMMDD may be useful for the storage of date values it is not how human beings commonly write or read dates. A person is more likely to be familiar with using dates that are in MMDDYY format. That is, a person is much more likely to be comfortable writing the date December 25, 2001 as "12/25/01" than "20011225."

Fortunately, it is a simple matter to code a routine that can be inserted between the user interface or data entry routines and the data storage routines that read from or write to magnetic disk. Thus, date values can be saved on disk in whatever format is deemed convenient for storage and sorting while at the same time being presented in the user interface, data entry routines, and printed reports in whatever format is deemed convenient and familiar for human users.

14.4 DATA ENTRY METHODS

•keyboards
•optical character recognition (OCR)
•magnetic ink character recognition (MICR)
•mark-sense forms
•punch-out forms
•bar codes
•intelligent terminals

Tests for validating input data include: test for missing data, test for correct field length, test for class or composition, test for range or reasonableness, test for invalid values, test for comparison with stored data, setting up self-validating codes, and using check digits. Tests for class or composition are used to check whether data fields are correctly filled in with either numbers or letters. Tests for range or reasonableness do not permit a user to input a date such as October 32.

This is sometimes called a sanity check.

Database

A database is a group of related files. This collection is usually organized to facilitate efficient and accurate inquiry and update. A database management system (DBMS) is a software package that is used to organize and maintain a database.

Usually when we use the word "file" we mean traditional or conventional files. Sometimes we call them "flat files." With these traditional, flat files each file is a single, recognizable, distinct entity on your hard disk. These are the kind of files that you can see cataloged in your directory. Commonly, these days, when we use the word "database" we are not talking about a collection of this kind of file; rather we would usually be understood to be talking about a database management system. And, commonly, people who work in a DBMS environment speak in terms of "tables" rather than "files." DBMS software allows data and file relationships to be created, maintained, and reported. A DBMS offers a number of advantages over file-oriented systems including reduced data duplication, easier reporting, improved security, and more rapid development of new applications. The DBMS may or may not store a table as an individual, distinct disk file. The software may choose to store more than one table in a single disk file. Or it may choose to store one table across several distinct disk files, or even spread it across multiple hard disks. The details of physical storage of the data are not important to the end user who only is concerned about the logical tables, not physical disk files.

In a hierarchical database the data is organized in a tree structure. Each parent record may have multiple child records, but any child may only have one parent. The parent-child relationships are established when the database is first generated, which makes later modification more difficult.

A network database is similar to a hierarchical database except that a child record (called a "member") may have more than one parent (called an "owner"). Like in a hierarchical database, the parent-child relationships must be defined before the database is put into use, and the addition or modification of fields requires the relationships to be redefined.

In a relational database the data is organized in tables that are called "relations." Tables are usually depicted as a grid of rows ("tuples") and columns ("attributes"). Each row is a record; each column is a field. With a relational database links between tables can be established at any time provided the tables have a field in common. This allows for a great amount of flexibility.

14.5 SYSTEM IMPLEMENTATION

Systems implementation is the construction of the new system and its delivery into ‘production’ or day-to-day operation.

The key to understanding the implementation phase is to realize that there is a lot more to be done than programming. During implementation you bring your process, data, and network models to life with technology. This requires programming, but it also requires database creation and population, and network installation and testing. You also need to make sure the people are taken care of with effective training and documentation. Finally, if you expect your development skills to improve over time, you need to conduct a review of the lessons learned.

During both design and implementation, you ought to be looking ahead to the support phase. Over the long run, this is where most of the costs of an application reside.

Systems implementation involves installation and changeover from the previous system to the new one, including training users and making adjustments to the system. Many problems can arise at this stage. You have to be extremely careful in implementing new systems. First, users are probably nervous about the change already. If something goes wrong they may never trust the new system. Second, if major errors occur, you could lose important business data.

A crucial stage in implementation is final testing. Testing and quality control must be performed at every stage of development, but a final systems test is needed before staff entrust the company's data to the new system. Occasionally, small problems will be noted, but their resolution will be left for later.

In any large system, errors and changes will occur, the key is to identify them and determine which ones must be fixed immediately. Smaller problems are often left to the software maintenance staff. Change is an important part of MIS. Designing and implementing new systems often causes changes in the business operations. Yet, many people do, not like changes. Changes require learning new methods, forging new relationships with people and managers, or perhaps even loss of jobs. Changes exist on many levels: in society, in business, and in information systems. Changes can occur because of shifts in the environment, or they can be introduced by internal change agents. Left to themselves, most organizations will resist even small changes. Change agents are objects or people who cause or facilitate changes. Sometimes it might be a new employee who brings fresh ideas; other times changes can be mandated by top-level management. Sometimes an outside event such as arrival of a new competitor or a natural disaster forces an organization to change. Whatever the cause, people tend to resist change.

However, if organizations do not change, they cannot survive. The goal is to implement systems in a manner that recognizes resistance to change but encourages people to accept the new system. Effective implementation involves finding ways to reduce this resistance. Sometimes, implementation involves the cooperation of outsiders such as suppliers.

Because implementation is so important, several techniques have been developed to help implement new systems. Direct cutover is an obvious technique, where the old system is simply dropped and the new one started. If at all possible, it is best to avoid this technique, because it is the most dangerous to data. If anything goes wrong with the new system, you run the risk of losing valuable information because the old system is not available.

In many ways, the safest choice is to use parallel implementation. In this case, the new system is introduced alongside the old one. Both systems are operated at the same time until you determine that the new system is acceptable. The main drawback to this method is that it can be expensive because data has to be entered twice. Additionally, if users are nervous about the new system, they might avoid the change and stick with the old method. In this case, the new system may never get a fair trial.

If you design a system for a chain of retail stores, you could pilot test the first implementation in one store. By working with one store at a time, there are likely to be fewer problems. But if problems do arise, you will have more staff members around to overcome the obstacles. When the system is working well in one store, you can move to the next location. Similarly, even if there is only one store, you might be able to split the implementation into sections based on the area of business. You might install a set of computer cash registers first. When they work correctly, you can connect them to a central computer and produce daily reports. Next, you can move on to annual summaries and payroll. Eventually the entire system will be installed.

Let us now see the Process of Implementation which involves the following steps:

- Internal or outsourcing (trend is "outsourcing")
- Acquisition: purchasing software, hardware, etc.
- Training: employee (end-users) training, technical staff training. SQL training in 5 days costs around $2000, + airplane, hotel, meals, rental car ($3000 to 5000); evaluation
- Testing:
- a bigger system requires more testing time
- a good career opportunity for non-technical people who wish to get in the door in the IT jobs.
- Documentation:
- backup
- knowledge management system
- Actual Installation
- Conversion: Migration from the old system to a new system
- Maintenance: very important; if you don't maintain the new system properly, it's useless to develop a new system.
- monitor the system,

- Upgrade,
- Trouble-shooting,
- Continuous improvement

14.6 SYSTEM MAINTENANCE

Once the system is installed, the MIS job has just begun. Computer systems are constantly changing. Hardware upgrades occur continually, and commercial software tools may change every year. Users change jobs. Errors may exist in the system. The business changes, and management and users demand new information and expansions. All of these actions mean the system needs to be modified. The job of overseeing and making these modifications is called software maintenance.

The pressures for change are so great that in most organizations today as much as 80 per cent of the MIS staff is devoted to modifying existing programs. These changes can be time consuming and difficult. Most major systems were created by teams of programmers and analysts over a long period. In order to make a change to a program, the programmer has to understand how the current program works.

Because the program was written by many different people with varying styles, it can be hard to understand. Finally, when a programmer makes a minor change in one location, it can affect another area of the program, which can cause additional errors or necessitate more changes.

One difficulty with software maintenance is that every time part of an application is modified, there is a risk of adding defects (bugs). Also, over time the application becomes less structured and more complex, making it harder to understand. These are some of the main reasons why the year 2000 alterations were so expensive and time consuming. At some point, a company may decide to replace or improve the heavily modified system. There are several techniques for improving an existing system, ranging from rewriting individual sections to restructuring the entire application.. The difference lies in scope-how much of the application needs to be modified. Older applications that were subject to modifications over several years tend to contain code that is no longer used, poorly documented changes, and inconsistent naming conventions. These applications are prime candidates for restructuring, during which the entire code is analyzed and reorganized to make it more efficient. More important, the code is organized, standardized, and documented to make it easier to make changes in the future.

14.7 SYSTEM EVALUATION

An important phase in any project is evaluating the resulting system. As part of this evaluation, it is also important to assess the effectiveness of the particular development process. There are several questions to ask. Were the initial cost estimates accurate? Was the project completed on time? Did users have sufficient input? Are maintenance costs higher than expected?

Evaluation is a difficult issue. How can you as a manager tell the difference between a good system and a poor one? In some way, the system should decrease costs, increase revenue, or provide a competitive advantage. Although these effects are important, they are often subtle and difficult to measure. The system should also be easy to use and flexible enough to adapt to changes in the business. If employees or customers continue to complain about a system, it should be re-examined.

A system also needs to be reliable. It should be available when needed and should produce accurate output. Error detection can be provided in the system to recognize and avoid common problems. Similarly, some systems can be built to tolerate errors, so that when errors arise, the system recognizes the problem and works around it. For example, some computers exist today that automatically switch to backup components when one section fails, thereby exhibiting fault tolerance.

Managers concern to remember when dealing with new systems is that the evaluation mechanism should be determined at the start. The question of evaluation is ignored until someone questions the value of the finished product. It is a good design practice to ask what would make this system a good system when it is finished or how we can tell a good system from a bad one in this application. Even though these questions may be difficult to answer, they need to be asked. The answers, however incomplete, will provide valuable guidance during the design stage.

Recall that every system needs a goal, a way of measuring progress toward that goal, and a feedback mechanism. Traditionally, control of systems has been the task of the computer programming staff. Their primary goal was to create error-free code, and they used various testing techniques to find and correct errors in the code. Today, creating error-free code is not a sufficient goal. We have all heard the phrase, "The customer is always right." The meaning behind this phrase is that sometimes people have different opinions on whether a system is behaving correctly. When there is a conflict, the opinion that is most important is that of the customer. In the final analysis, customers are in control because they can always take their business elsewhere. With information systems, the users are the customers and the users should be the ones in control. Users determine whether a system is good. If the users are not convinced that the system performs useful tasks, it is not a good system.

Feasibility comparison Cost and budget Compare actual costs to budget estimates Time estimates Revenue effects Was project completed on time?

Maintenance costs Does system produce additional revenue?

Project goals how much money and time are spent on changes? Does system meet the initial goals of the project?

User satisfaction how do users (and management) evaluate the system?

System performance

System reliability: Are the results accurate and on time?
System availability: Is the system available continually?
System security: Does the system provides access to authorized users?

Summary: In this chapter, we learned Data Format, Data Entry and Data Storage, Date Formats, Data Entry Methods, System Implementation, System Maintenance, System Evaluation.

Questions:
1. Explain System Implementation in detail?
Ans: refer 14.5

2. Explain System Maintenance in detail?
Ans: refer 14.6

3. Explain System Evaluation?
Ans: refer 14.7

❖❖❖❖

15

DOCUMENTATION

Unit Structure

15.1 INTRODUCTION:

Documentation is an important part of software engineering. Types of documentation include:

Requirements - Statements that identify attributes capabilities, characteristics, or qualities of a system. This is the foundation for what shall be or has been implemented.

1. Architecture/Design - Overview of software. Includes relations to an environment and construction principles to be used in design of software components.
2. Technical - Documentation of code, algorithms, interfaces, and APIs.
3. End User - Manuals for the end-user, system administrators and support staff.
4. Marketing - How to market the product and analysis of the market demand.

15.2 REQUIREMENTS DOCUMENTATION

Requirements documentation is the description of what particular software does or shall do. It is used throughout development to communicate what the software does or shall do. It is also used as an agreement or as the foundation for agreement on what the software shall do. Requirements are produced and consumed by everyone involved in the production of software: end users, customers, product managers, project managers, sales, marketing, software architects, usability experts, interaction designers, developers, and testers, to name a few. Thus, requirements documentation has many different purposes.

Requirements come in a variety of styles, notations and formality. Requirements can be goal-like (e.g., *distributed work environment*), close to design (e.g., *builds can be started by right-clicking a configuration file and select the 'build' function*), and anything in between. They can be specified as statements in natural language, as drawn figures, as detailed mathematical formulas, and as a combination of them all.

The variation and complexity of requirements documentation makes it a proven challenge. Requirements may be implicit and hard to uncover. It is difficult to know exactly how much and what kind of documentation is needed and how much can be left to the architecture and design documentation, and it is difficult to know how to document requirements considering the variety of people that shall read and use the documentation. Thus, requirements documentation is often incomplete (or non-existent). Without proper requirements documentation, software changes become more difficult—and therefore more error prone (decreased software quality) and time-consuming (expensive).

The need for requirements documentation is typically related to the complexity of the product, the impact of the product, and the life expectancy of the software. If the software is very complex or developed by many people (e.g., mobile phone software), requirements can help to better communicate what to achieve. If the software is safety-critical and can have negative impact on human life (e.g., nuclear power systems, medical equipment), more formal requirements documentation is often required. If the software is expected to live for only a month or two (e.g., very small mobile phone applications developed specifically for a certain campaign) very little requirements documentation may be needed. If the software is a first release that is later built upon, requirements documentation is very helpful when managing the change of the software and verifying that nothing has been broken in the software when it is modified.

Traditionally, requirements are specified in requirements documents (e.g. using word processing applications and spreadsheet applications). To manage the increased complexity and changing nature of requirements documentation (and software documentation in general), database-centric systems and special-purpose requirements management tools are advocated.

15.3 ARCHITECTURE/DESIGN DOCUMENTATION

Architecture documentation is a special breed of design document. In a way, architecture documents are third derivative from the code (design document being second derivative, and code documents being first). Very little in the architecture documents is specific to the code itself. These documents do not describe how to program a particular routine, or even why that particular routine exists in the form that it does, but instead merely lays out the general requirements that would motivate the existence of such a routine. A good architecture document is short on details but thick on explanation. It may suggest approaches for lower level design, but leave the actual exploration trade studies to other documents.

Another breed of design docs is the comparison document, or trade study. This would often take the form of a *whitepaper*. It focuses on one specific aspect of the system and suggests alternate approaches. It could be at the user interface, code, design, or even architectural level. It will outline what the situation is, describe one or more alternatives, and enumerate the pros and cons of each. A good trade study document is heavy on research, expresses its idea clearly (without relying heavily on obtuse jargon to dazzle the reader), and most importantly is impartial. It should honestly and clearly explain the costs of whatever solution it offers as best. The objective of a trade study is to devise the best solution, rather than to push a particular point of view. It is perfectly acceptable to state no conclusion, or to conclude that none of the alternatives are sufficiently better than the baseline to warrant a change. It should be approached as a scientific endeavour, not as a marketing technique.

A very important part of the design document in enterprise software development is the Database Design Document (DDD). It contains Conceptual, Logical, and Physical Design Elements. The DDD includes the formal information that the people who interact with the database need. The purpose of preparing it is to create a common source to be used by all players within the scene. The potential users are:

- Database Designer
- Database Developer
- Database Administrator
- Application Designer
- Application Developer

When talking about Relational Database Systems, the document should include following parts:

- Entity - Relationship Schema, including following information and their clear definitions:
    - Entity Sets and their attributes
    - Relationships and their attributes
    - Candidate keys for each entity set
    - Attribute and Tuple based constraints
- Relational Schema, including following information:
    - Tables, Attributes, and their properties
    - Views
    - Constraints such as primary keys, foreign keys,
    - Cardinality of referential constraints
    - Cascading Policy for referential constraints
    - Primary keys

It is very important to include all information that is to be used by all actors in the scene. It is also very important to update the documents as any change occurs in the database as well.

15.4 TECHNICAL DOCUMENTATION

This is what most programmers mean when using the term *software documentation*. When creating software, code alone is insufficient. There must be some text along with it to describe various aspects of its intended operation. It is important for the code documents to be thorough, but not so verbose that it becomes difficult to maintain them. Several How-to and overview documentation are found specific to the software application or software product being documented by API Writers. This documentation may be used by developers, testers and also the end customers or clients using this software application. Today, we see lot of high end applications in the field of power, energy, transportation, networks, aerospace, safety, security, industry automation and a variety of other domains. Technical documentation has become important within such organizations as the basic and advanced level of information may change over a period of time with architecture changes. Hence, technical documentation has gained lot of importance in recent times, especially in the software field.

Often, tools such as Doxygen, NDoc, javadoc, EiffelStudio, Sandcastle, ROBODoc, POD, TwinText, or Universal Report can be used to auto-generate the code documents—that is, they extract the comments and software contracts, where available, from the source code and create reference manuals in such forms as text or

HTML files. Code documents are often organized into a *reference guide* style, allowing a programmer to quickly look up an arbitrary function or class.

Many programmers really like the idea of auto-generating documentation for various reasons. For example, because it is extracted from the source code itself (for example, through comments), the programmer can write it while referring to the code, and use the same tools used to create the source code to make the documentation. This makes it much easier to keep the documentation up-to-date.

Of course, a downside is that only programmers can edit this kind of documentation, and it depends on them to refresh the output (for example, by running a job to update the documents nightly). Some would characterize this as a pro rather than a con.
Donald Knuth has insisted on the fact that documentation can be a very difficult afterthought process and has advocated literate programming, writing at the same time and location as the source code and extracted by automatic means.

Elucidative Programming is the result of practical applications of Literate Programming in real programming contexts. The Elucidative paradigm proposes that source code and documentation be stored separately. This paradigm was inspired by the same experimental findings that produced Kelp. Often, software developers need to be able to create and access information that is not going to be part of the source file itself. Such annotations are usually part of several software development activities, such as code walks and porting, where third party source code is analysed in a functional way. Annotations can therefore help the developer during any stage of software development where a formal documentation system would hinder progress. Kelp stores annotations in separate files, linking the information to the source code dynamically.

15.5 USER DOCUMENTATION

Unlike code documents, user documents are usually far more diverse with respect to the source code of the program, and instead simply describe how it is used.

In the case of a software library, the code documents and user documents could be effectively equivalent and are worth conjoining, but for a general application this is not often true. On the other hand, the Lisp machine grew out of a tradition in which every piece of code had an attached documentation string. In combination with strong search capabilities (based on a Unix-like *apropos* command), and online sources, Lisp users could look up documentation prepared by these API Writers and paste the associated function directly into their own code. This level of ease of use is unheard of in putatively more modern systems.

Typically, the user documentation describes each feature of the program, and assists the user in realizing these features. A good user document can also go so far as to provide thorough troubleshooting assistance. It is very important for user documents to not be confusing, and for them to be up to date. User documents need not be organized in any particular way, but it is very important for them to have a thorough index. Consistency and simplicity are also very valuable. User documentation is considered to constitute a contract specifying what the software will do. API Writers are very well accomplished towards writing good user documents as they would be well aware of the software architecture and programming techniques used. See also Technical Writing.

There are three broad ways in which user documentation can be organized.

1. **Tutorial:** A tutorial approach is considered the most useful for a new user, in which they are guided through each step of accomplishing particular tasks .
2. **Thematic:** A thematic approach, where chapters or sections concentrate on one particular area of interest, is of more general use to an intermediate user. Some authors prefer to convey their ideas through a knowledge based article to facilitating the user needs. This approach is usually practiced by a dynamic industry, such as Information technology, where the user population is largely correlated with the troubleshooting demands.
3. **List or Reference:** The final type of organizing principle is one in which commands or tasks are simply listed alphabetically or logically grouped, often via cross-referenced indexes. This latter approach is of greater use to advanced users who know exactly what sort of information they are looking for.

A common complaint among users regarding software documentation is that only one of these three approaches was taken to the near-exclusion of the other two. It is common to limit provided software documentation for personal computers to online help that give only reference information on commands or menu items. The job of tutoring new users or helping more experienced users get the most out of a program is left to private publishers, who are often given significant assistance by the software developer.

15.6 MARKETING DOCUMENTATION

For many applications it is necessary to have some promotional materials to encourage casual observers to spend more time learning about the product. This form of documentation has three purposes:-

1. To excite the potential user about the product and instil in them a desire for becoming more involved with it.
2. To inform them about what exactly the product does, so that their expectations are in line with what they will be receiving.
3. To explain the position of this product with respect to other alternatives.
4. To completely shroud the function of the product in mystery.

One good marketing technique is to provide clear and memorable *catch phrases* that exemplify the point we wish to convey, and also emphasize the interoperability of the program with anything else provided by the manufacturer.

15.7 CASE TOOLS AND THEIR IMPORTANCE

CASE tools stand for **C**omputer **A**ided **S**oftware **E**ngineering tools. As the name implies they are computer based programs to increase the productivity of analysts. They permit effective communication with users as well as other members of the development team. They integrate the development done during each phase of a system life cycle and also assist in correctly assessing the effects and cost of changes so that maintenance cost can be estimated.

- **Available CASE tools**

Commercially available systems provide tools (i.e. computer program packages) for each phase of the system development life cycle. A typical package is Visual Analyst which has several tools integrated together. Tools are also in the open domain which can be downloaded and used. However, they do not usually have very good user interfaces.

Following types of tools are available:

- System requirements specification documentation tool
- Data flow diagramming tool
- System flow chart generation tool
- Data dictionary creation
- Formatting and checking structured English process logic
- Decision table checking

 Screen design for data inputting
 Form design for outputs.
 E-R diagramming
 Data base normalization given the dependency information

- **When are tools used**

Tools are used throughout the system design phase. CASE tools are sometimes classified as upper CASE tools and lower CASE tools. The tools we have described so far are upper CASE tools.

They are tools which will generate computer screen code from higher level descriptions such as structured English and decision tables, such tools are called lower CASE tools

- **Object Oriented System Design Tools**

Unified Modelling Language is currently the standard. UML tool set is marketed by Rational Rose a company whose tools are widely used. This is an expensive tool and not in our scope in his course.

- **How to use the tools**

•Most tools have a user’s guide which is given as help files along with the tool

•Many have FAQ’s and search capabilities

•Details on several open domain tools and what they do is given below.

- **System Flowchart and ER-Diagram generation Tool**

Name of the tool: SMARTDRAW

URL: This Software can be downloaded from: http://www.smartdraw.com. This is a paid software, but a 30-day free trial for learning can be downloaded.

Requirements to use the tool: PC running Windows 95, 98 or NT. The latest versions of Internet Explorer or Netscape Navigator, and about 20MB of free space.

What the tool does: Smartdraw is a perfect suite for drawing all kinds of diagrams and charts: Flowcharts, Organizational charts, Gantt charts, Network diagrams, ERdiagrams etc.

The drag and drop readymade graphics of thousands of templates from built-in libraries makes drawing easier. It has a large drawing area and drawings from this tool can be embedded into Word, Excel and PowerPoint by simply copy-pasting. It has an extensive collection of symbols for all kinds of drawings.

How to use: The built-in tips guides as the drawing is being created. Tool tips automatically label buttons on the tool bar.
There is online tutorial provided in:
http://www.smartdraw.com/tutorials/flowcharts/tutorials1.htm
http://www.ttp.co.uk/abtsd.html

- **Data Flow Diagram Tool**

Name of the tool: IBMS/DFD
URL: This a free software that can be downloaded from: http://viu.eng.rpi.edu

Requirements to use the tool: The following installation instructions assume that the user uses a PC running Windows 95, 98 or NT. Additionally, the instructions assume the use of the latest versions of Internet Explorer or Netscape Navigator. To download the zip files & extract them you will need WinZip or similar software. If needed download at http://www.winzip.com.

What the tool does: The tool helps the users draw a standard data flow diagram (a process-oriented model of information systems) for systems analysis.

How to use: Double click on the IBMS icon to see the welcome screen. Click Any where inside the welcome screen to bring up the first screen.

Under "Tools" menu, select DFD Modelling. The IBMS will pop up the Data Flow Diagram window. Its menu bar has the File, Edit, Insert, Font, Tool, Window and Help options. Its tool box on the right contains 10 icons, representing (from left to right and top to bottom) pointer, cut, data flow, process, external entity, data store, zoom-out, zoom-in, decompose, and compose operations, respectively.

Left click on the DFD component to be used in the toolbox, key in the information pertaining to it in the input dialogue box that prompts for information.

To move the DFD components: Left click on the Pointer icon in the tool box, point to the component, and hold Left Button to move to the new location desired in the work area.

To edit information of the DFD components: Right click on the DFD component. The input dialogue box will prompt you to edit information of that component.

Levelling of DFD: Use the Decompose icon in the tool box for levelling.

To save the DFD: Under File menu, choose Save or SaveAs. Input the name and extension of the DFD (the default extension is DFD) and specify folder for the DFD to be saved. Click OK.

- **System requirement specification documentation tool**

Name of the tool: ARM
URL: The tool can be downloaded without cost at
http://sw-assurance.gsfc.nasa.gov/disciplines/quality/index.php

What the tool does: ARM or Automated Requirement Measurement tool aids in writing the System Requirements Specifications right. The user writes the SRS in a text file, the ARM tool scans this file that contains the requirement specifications and gives a report file with the same prefix name as the user's source file and adds an extension of ".arm". This report file contains a category called INCOMPLETE that indicate the words and phrases that are not fully developed.

Requirements to use the tool: PC running Windows 95, 98 or NT. The latest versions of Internet Explorer or Netscape Navigator, and about 8MB of free space.

How to use the tool : On clicking the option Analyze under File menu and selecting the file that contains the System Requirements Specifications, the tool processes the document to check if the specifications are right and generate a ARM report.

The WALKTHROUGH option in the ARM tool assists a user by guiding him as to how to use the tool apart from the HELP menu. The README.doc file downloaded during installation also contains description of the usage of this tool.

- **A Tool for designing and Manipulating Decision tables**

Name of the tool: Prologa V.5
URL: http://www.econ.kuleuven.ac.be/prologa
Note: This tool can be downloaded from the above given URL, after obtaining the password.

What the tool does: The purpose of the tool is to allow the decision maker to construct and manipulate (systems of) decision tables. In this construction process, the features available are automatic table contraction, automatic table optimization, (automatic) decomposition and composition of tables, verification and validation of tables and between tables, visual development, and rule based specification.

15.8 SUMMARY:

In this Chapter, we learned the concept of Documentation and types of documentation such as Requirements documentation Architecture/Design documentation, Technical documentation, User documentation, Marketing documentation and CASE Tools and their importance.

Questions:

1. Explain Requirements documentation?

Ans: refer 15.2

2. Explain Architecture/Design documentation?

Ans: refer 15.3

3. Explain Technical documentation?

Ans: refer 15.4

4. Explain User documentation?

Ans: refer 15.5

5. Explain Marketing documentation?

Ans: refer 15.6

❖❖❖❖