Kavayitri Bahinabai Chaudhari
North Maharashtra University

SEM. IV ▪ BCA 401

BCA

Bachelor in Computer Application

Software Engineering

▪ Dr. B. H. Barhate ▪ Sunil D. Mone
▪ Mahesh K. Bhavsar

Prashant

As per U.G.C. Guidelines and also on the basis of revised syllabus of
Kavayitri Bahinabai Chaudhari North Maharashtra University
with effect from Academic Year 2023-24. Also useful for all Universities.

SEM IV ▪ BCA-401

Bachelor in Computer Application

SOFTWARE ENGINEERING

Text Book Development Committee

- C O O R D I N A T O R -

Dr. Rakesh S. Deore
Head, Department of Computer Science,
S.S.V.P.S. Sanstahs Arts, Science and Commerce College, Dhule.

- A U T H O R S -

Dr. B. H. Barhate
Vice-Principal,
Bhusawal Arts, Commerce and P.O. Nahta Science College, Bhusawal.

Sunil D. Mone
Head, Department of Computer Science,
R. C. Patel Educational Trust's R. C. Patel Arts, Science and Commerce, Chalisgaon.

Mahesh K. Bhavsar
Assistant Professor,
R. C. Patel Educational Trust's Institute of Management Research and Development,
Shirpur.

Prashant
Prashant Publications

SOFTWARE ENGINEERING
SEM IV ▪ BCA-401

Publisher | Printer
Prashant Publications
3, Pratap Nagar, Sant Dnyaneshwar Mandir Rd,
Near Nutan Maratha College, Jalgaon 425001.

Phone | Website | E-mail
0257-2235520, 0257-2232800, 9665626717
www.prashantpublications.com
prashantpublication.jal@gmail.com

Distributor
Prashant Book House
17, Stadium Shopping Centre, Opp. SBI, Jalgaon.
Ph. 0257-2227231

Available at all leading booksellers
Jalgaon | Dhule | Nadurbar

First Edition, January, 2024

ISBN 978-81-19120-92-5

Price ₹ 60/-

PREFACE

SOFTWARE ENGINEERING is a Simple version for S.Y.B.C.A. students, published by Prashant Publication.

This text is in accordance with the new syllabus CBCS-2023 recommended by the Kavayitri Bahainabai Chaudhari North Maharashtra University, Jalgaon, which has been serving the need of S.Y.B.C.A. Computer Science students from various colleges. This text is also useful for the student of Engineering, B.Sc. (Information Technology and Computer Science), M.Sc, M.C.A. B.B.M., M.B.M. other different Computer courses.

We are extremely grateful to Prof. Dr. S. R. Kolhe, Professor, Director, School of Computer Sciences and Chairman, Board of Studies, Kavayitri Bahinabai Chaudhari North Maharashtra University, Jalgaon for his valuable guidance.

We are obligated to Principals Dr. D. R. Patil, Principal, R. C. Patel Arts, Science and Commerce College, Shirpur, Dr. Vaishali Patil, Director, R. C. Patels Institute of Management and Research, Shirpur, Dr. B. H. Barhate, Vice-principal, P. O. Nahta College, Bhusawal and Librarians and staff of respective colleges for their encouragement.

We are very much thankful to Shri. Rangrao Patil, Shri. Pradeep Patil of Prashant Publications, who has shown extreme co-operation during the preparation of this book, for getting the book published in time and providing an opportunity to be a part of this book.

We welcome any suggestions aimed at enhancing the content of this book, and we look forward to constructive feedback from our readers.

- Authors

S Y L L A B U S

Bachelor in Computer Applications (BCA)

Software Engineering | Sem. IV | BCA-401

w.e.f. Academic Year 2023-24 (Semester System 60 + 40 Pattern)

Unit 1 : System Concept and Information **(10 L, 15 M)**

- Definition and Characteristics of System
- Elements of Systems
- Types of system – Conceptual, Physical, Natural, Artificial, Open & Closed, Deterministic etc.

Unit 2 : System Development Life Cycle **(10 L, 15 M)**

- Roles of System Analysts- As an Architect, Change Agent, Investigator & Monitor, Organizer, Motivator etc.
- Introduction of Systems Development Life Cycle (SDLC)
- SDLC Models :
  - Waterfall Model
  - Spiral Model
  - RAD Model
  - Prototyping Model

Unit 3 : System Planning **(10 L, 15 M)**

- Data and fact gathering techniques: Interviews & Questionnaires, Group discussion, On-site observation, Review of Written Documents.
- Introduction to Feasibility Study
- Types of feasibility study - Technical, Economical and Operational
- Introduction to SRS (Software Requirement Specifications)
- Need of SRS

Unit 4 : Systems Design and Modelling **(10 L, 15 M)**

- Computer Aided Software Engineering (CASE)
  - Systems Flowcharts
  - Data Flow Diagrams
  - Entity Relationship Diagram
- Tools for Structured Analysis :
  - Decision Tree
  - Decision Tables
  - Structured English

Unit 5 : User Interface of System & Software Testing **(10 L, 15 M)**

- User - Interface Design
- Graphical interfaces
- Elements of Good Interface Design
- Introduction to Software Testing
- Introduction to Black-Box and White Box Testing

Unit 6 : Designing Business Application System using DFD, ERD **(10 L, 15 M)**

- Library Management System
- Inventory Management System
- Hospital Management System
- Sales/Purchase System

CONTENTS

1. Chapter

System Concept and Information

C o n t e n t s :

(10 L, 15 M)

What is System :

- The term system is derived from the Greek word **system a**, which means an organized relationship among functioning units or components.
- We come into daily contact with the **transportation system**, the **telephone system**, the **accounting system**, the **production system**, **computer system** etc.
- Similarly, we talk of the **business system** and of the organization as a system consisting of interrelated departments (subsystems) such as **production**, **sales**, **personnel**, and an **information system**.
- **For Example -** may be single computer with a keyboard, memory, and printer or a series of intelligent terminals linked to a mainframe.
- Each component is part of the total system and has to do its share of work for the system to achieve the intended goal.
- This orientation requires an orderly grouping of the components for the design of a successful system.

1.1 Definition :

"A system is an orderly grouping of interdependent components linked together according to a plan to achieve a specific objective."

1.2 Characteristics of System :

Organization :

- Organization implies structure and order.
- It can also be defined as the **arrangement of components** that helps to **achieve objectives**.
- **Example :** In the design of a business system, the hierarchical relationships starting with the president on top and leading towards the workers represents the organization structure. So this gives the authority structure and specifies the formal flow of communication.
- In a **computer system** is designed around an **input device, a central processing unit, an output device and one or more storage units**.

Interaction :

- Interaction refers to the manner in which **each component functions with other components of the system**.ie, there should be an **interrelationship between each components of a system**.
- **Example**: In an organization there should be interaction between purchase department and production department, same way advertising with sales etc.
- **In computer system**, the **central processing unit must interact with the input device** to solve a problem. In turn the main memory holds programs and data that the arithmetic unit uses for computation.

Interdependence :

- Interdependence means the **parts or the components of an organization or computer system depends on one another**.
- Each component or parts should depend on other components of an organization.
- **One component or subsystem depends on the input of another subsystem for proper functioning**,
- **Output of one subsystem is required input for another subsystem**.
- **For example:** A decision to computerize an application is initiated by the user, analyzed and designed by the analyst, programmed and tested by the computer operator.

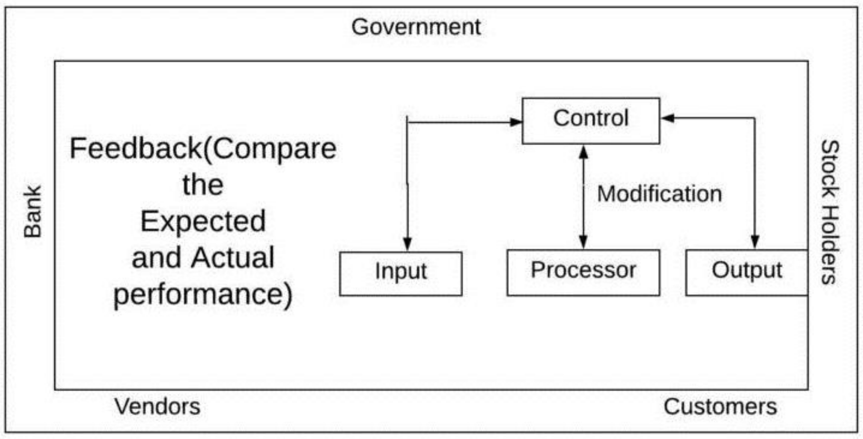
Integration :

- It is concerned with **how a system is tied together**.
- Integration is concerned with how a system is linked together.
- It is more than sharing a physical part or location.
- It means that **parts of the system work together within the system even though each part performs a unique function**.

Central Objective :

- A system should have a central objective.
- The important point is that users must know the central objective of computer application early in the analysis phase for a successful design and conversion.

1.3 Elements of System :



1. **Output and Input**
2. **Process**
3. **Control**
4. **Feedback**
5. **Environment**
6. **Boundaries and Interfaces**

1. **Output and Input :**
   - Inputs are the information or elements that we enter the system for processing.
   - A **system feeds on input to produce output**. Output is the **result of processing**.
   - A major objective of a system is to produce an output that has value to its user.
   - We must determine what the objectives or goals are, what we to achieve. Once we know our aim, we can try to achieve it in the best possible way.
2. **Processer :**
   - The processor is the element of a system that involves the actual transformations of input into output.
   - The **processor is the operational component of a system**.
   - This involves the programs and the way in which **data is processed through the computer**.

- **Processors may modify the input totally or partially**, depending on the specifications of the output.

3. **Control :**
   - Control of the system is the decision-maker that **controls the activities of accepting input processing and producing output.**
   - It is the decision-making subsystem that controls the pattern of activities, input, processing and output.
   - Control elements are the logical procedures, rules and regulations which direct and manage the processing of the inputs in order to produce the desired outputs.
   - The inputs and processors are checked, processed, managed and administer by the control elements.
4. **Feedback :**
   - Feedback is the measurement of outputs against some set of standard benchmarks.
   - Output is compared against performance standards, changes can result in the input or processing and consequently, the output.
   - Feedback may be positive or negative, routine or informational.
   - Feedback can be positive or negative. Positive feedback reinforces the performance of the system.
   - Negative feedback provides some information for action that will help us to improve the quality of the output.
5. **Environment :**
   - The environment is the area where the organization operates.
   - An environment may consist of vendors, competitors etc.
   - The environment is the source of external elements that have an effect on the system. In fact, it determines how a system must function.
6. **Boundaries and interface :**
   - **Boundaries** : A system should be defined by its boundaries - the limits that identify its components, processes & interrelationships when interfaces with another system.
   - **Example** : A teller system in a commercial bank is restricted to the deposits, withdrawals & related activities of customers checking and savings accounts.
   - **Interface** : "Interface is the element which is helpful in the interaction between the system and environment'.

1.4 Types of System :

Physical or Abstract System :

- **Physical** systems are **tangible entities**. We can **touch** and **feel them**.
- Physical System may be static or dynamic in nature.
- **For example :** Table and chairs are the physical parts of computer center which are static. A programmed computer is a dynamic system in which programs, data, and applications can change according to the user's needs.
- **Abstract systems** are non-physical entities or **conceptual that may be formulas, representation or model of a real system**.

Open or Closed System :

- An **open system** must interact with its environment.
- It receives inputs from and delivers outputs to the outside of the system.
- For example, an information system which must adapt to the **changing environmental conditions**.
- A **closed system does not interact with its environment**. It is **isolated** from environmental influences. A completely closed system is **rare in reality**.

Adaptive and Non-Adaptive System :

- **Adaptive System responds to the change in the environment** in a way to improve their performance and to survive.
- **Example** : human beings, animals.
- **Non-Adaptive System** is the system which **does not respond** to the environment.
- **Example** : machines.

Permanent or Temporary System :

- **Permanent System** persists for **long time**. **Example** : business policies.
- **Temporary System** is made for **specified time** and after that they are demolished. **Example** : A DJ system is set up for a program and it is dissembled after the program.

Deterministic or Probabilistic System :

- **Deterministic system** operates in a **predictable** manner and the interaction between system components is known with **certainty**.
- **Example :** Two molecules of hydrogen and one molecule of oxygen makes water.
- **Probabilistic System** shows **uncertain** behavior. The **exact output is not known**.
- **Example :** Weather forecasting, mail delivery.

Natural and Manufactured System :

- **Natural systems** are created by the **nature**. **Example** : Solar system, seasonal system.
- **Manufactured System** is the **man-made system**. **Example** : Rockets, dams, trains.

Social, Human-Machine and Machine System :

- **Social** System is **made up of people**. **Example :** Social clubs, societies.
- In **Human-Machine** System, **both human and machines are involved** to perform a particular task. **Example :** Computer programming.
- **Machine** System is where **human interference is neglected**. All the **tasks** are **performed** by the **machine**. **Example :** Robot.

Man-Made Information System :

- **Formal Information System :** It is based on the flow of information in the form of memos, instructions, etc., from top level to lower levels of management.
- **Informal Information System :** This is employee based system which solves the day to day work related problems.
- **Computer Based System :** This system is directly dependent on the computer for managing business applications. For example, automatic library system, railway reservation system, banking system, etc.

: QUESTIONS :

1. What is System? Describe Characteristics of System.
2. Discuss different Elements of System.
3. Explain different Types of System.

***

2. Chapter

System Development Life Cycle

Contents:

(10 L, 15 M)

Introduction to System Analyst :

- **System analyst** is a person **responsible** for the **development** of **software** and **hardware** to the efficient working of the organization.
- Analysts study the **environment** and **problems** of an organization to decide whether a **new information method** can provide solution to the problem.
- Main job of system analyst is to ***provide right type of information***, ***in right quantity*** at the ***right time*** to the **management** or the **end user**.
- A systems analyst is an information technology **professional** who specializes in **analyzing**, **designing** and **implementing** information systems.
- Systems analysts assess the appropriate of information systems in terms of their **planned outcomes** and **link with end users**, **software vendors** and **programmers** in order to achieve these outcomes.
- A systems analyst is a person who uses **analysis** and **design techniques** to **solve business problems using information technology**.
- Systems analysts may serve as **change agents** who identify the organizational **improvements** needed, design systems to **implement** those **changes**, and **train** and **motivate others** to use the systems.

- Although they may be **familiar** with a **variety** of **programming languages**, **operating systems**, and computer **hardware platforms**.
- They may be responsible for developing **cost analysis, design, staff impact, and implementation timelines**.
- A systems analyst will often **evaluate** and **modify code** as well as **review scripting**.

Definition of System Analyst :

"A person who conducts a systematic study and evaluation of an activity such as a business to identify its desired objectives in order to decide procedures by which these objectives can be gained".

2.1 Roles of System Analyst :

1. **Architect :**
   - The architect's primary function as **connection between** the **clients abstract design request**.
   - System analysts coordinate between customers, IT persons, and stakeholders to develop information systems capable of delivering business requirements.
   - They have to design an information system architecture according to the user's requirements which acts as a blueprint for the programmers.
   - For that, they need to know exactly what users want and also have to build good relationships and understanding with them to understand their requirements as well as convey correct and complete information to the development team.
   - As architect the analyst also **creates** a **detailed physical design** of system.
2. **Analyst as Change Agent :**
   - The analyst may be viewed as an **agent** of **change**.
   - The candidate system is designed to introduce change and **re-orientation** in how the **user orientation handles** information or makes decision.
   - It is **important** that **change** be **accepted** by the user.
   - The way to **secure user acceptance** is through user participation during design and implementation.
   - The systems analyst may select **various styles** to **introduce a change** to the user organization.
   - **No matter what style is used**, the goal is the same : to achieve acceptance of the candidate system with a minimum of struggle.
   - As an analyst whenever perform any of the activities in the systems development life cycle and are **present** and **interacting** with **users** and the **business**.

- The person, who **changes**, **develops** a **plan** for **change**, and **works with others** in make easy that change.
- As a systems analyst, you must **identify** this **fact** and **use** it as a starting point for your analysis.
- Must interact with **users** and **management** from the **very beginning** of your project.
- Without their help you cannot understand what they need to support their work in the organization, and real change cannot take place.
- You also teach users the process of change, because changes in the information system **do not** occur independently.

3. Investigator & Monitor :

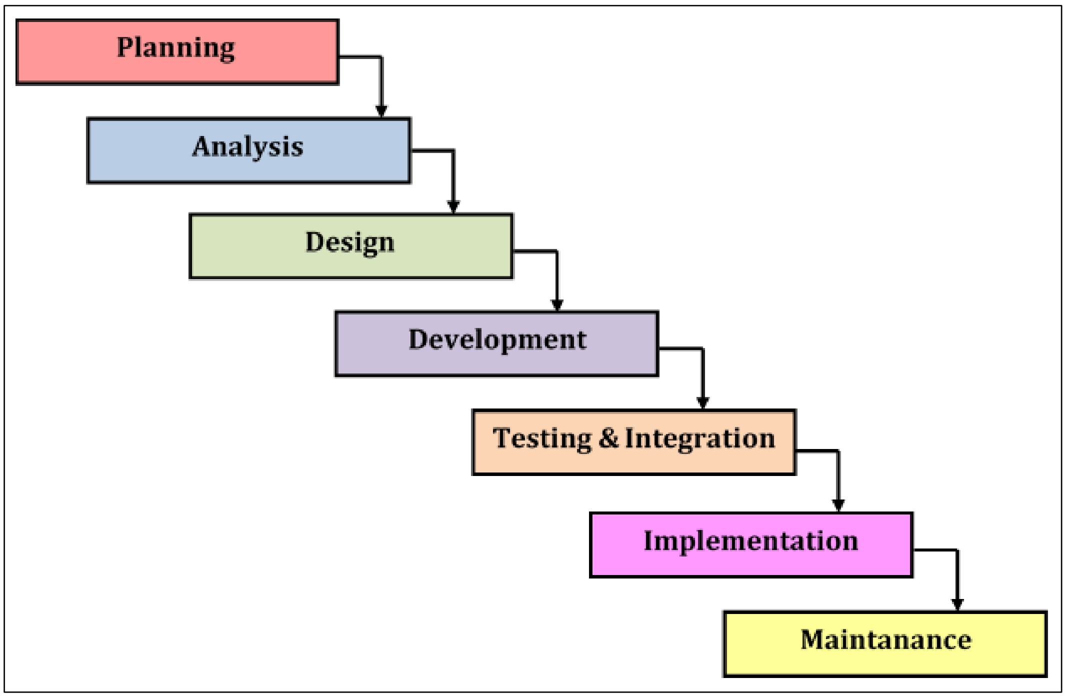
- In defining a problem, the analyst pieces together the **information gathered to determine why the present system does not work well** and **what changes will correct the problem**.
- System analyst work is **similar** to that of an **investigator**.
- Effective user participation and training with proper motivation to use the system are important factors to achieve system acceptance.
- Extracting the real problems from existing systems and creating information structures that uncover previously unknown trend that may have a direct impact on the organization.
- The analyst must monitor programs in **relation** to **time**, **cost** & **quality**.

4. Organizer :

- The basic and most important step for system analysts is to understand user's requirements clearly.
- This phase is important to understand how the current system functions and what users want from the new systems.
- System analysts act as researchers and gather various facts and data with the active cooperation from the users of the system.
- They consult users from time to time to obtain necessary information related to the system, and whether there is any last-minute requirement.
- This process is important because analysts have to organize and document information into functional specification to solve to develop a system.

5. **Monitor & Psychologist :**
   - In system development, **systems** are **building around** people.
   - Analyst plays the role of a psychologies in the way he/she **reaches people**, **interprets** their **thoughts** accesses their **behavior** and **draws conclusions** from these interactions.
   - Understanding inter-functional relationships is important.
   - It is important that the analyst be aware of people's **feelings** and be **prepared** to get **around** things in an **attractive** way.
   - The **art** of **listening** is **important** in **evaluating responses** and **feedback**.
6. **Qualification and Responsibilities :**
   - Through **education**, **experience** and **personality** the analyst acquire skills.
   - Most of the today's analysts are college graduates with majors in accounting, management or information system.
   - The background and experience of analyst includes -
     - ✓ A background in **systems theory** and **organization behaviour**.
     - ✓ Familiarity with the makeup and inner working of major application areas like financial accounting, personnel administration, marketing and sales, operations management, model building and production control.
     - ✓ Ability in system tools and methodologies and a **practical knowledge** of one or more **programming** and **data base languages**.
     - ✓ Experience in hardware and software specifications, which is important for selection.
     - ✓ **Authority** – The confidence to "tell" people what to do. Much of this quality shows in project management and team work to meet deadlines.
     - ✓ **Communication skills** – ability to articulate and focus on a problem area for logical solutions.
     - ✓ **Creativity –** Trying one's own ideas, developing candidate systems using unique tools or methods.

2.2 What is Systems Development Life Cycle (SDLC) :



1. **Planning** :
   - **Define** the **problem** and **scope** of **existing system**.
   - **Overview** the **new system** and **decide** its **objectives**.
   - Confirm project **feasibility** and **create** the **project Schedule**.
   - During this **phase**, **threats**, **constraints**, **integration** and **security** of system are also considered.
   - Planning is **starting phase** of the SDLC process that sets out to **discover**, **identify**, and **define** the scope of the project to **decide the course of action**.
   - During this phase that understand performed to **decide resources**, **budget**, **personnel**, **technical aspects**, **overall capabilities** and more.
2. **Analysis :**
   - The purpose of this phase is to understand the **business** and **processing needs** of the information system project.
   - The development team considers the **useful requirements** of the system to assess **how the solution will meet** the **end user's expectations**.
   - **Gather**, **analyze**, and **validate** the information.

- Define the **requirements** and **model** for new system.
- Evaluate the **alternatives** and **prioritize** the **requirements**.
- Examine the **information needs** of **end-user** and **improve** the **system goal**.
- A Software Requirement **document**, **software**, **hardware**, **functional**, and **network requirements** of the system.
- The end user requirements are determined the project is feasible from a **financial**, **organizational**, **social**, and **technological**.

3. **Design :**
   - After a complete analysis phase, the design phase will start.
   - The **elements**, **components**, **security levels**, **modules**, **architecture**, **interfaces** and **data of the system** are defined and designed to evaluate **how the finished system will work and what it will look like**.
   - The system design is produced in **detail** to ensure the system will include the necessary **features to meet all functional and operational aspects of the project**.
4. **Development :**
   - In this phase, the development team is hard at work writing code and constructing and fine-tuning the technical and physical configurations necessary to build the overall information system.
   - This is considered by many as the most robust phase in the life cycle as all the labor-intensive efforts are made here, signifying the real start of software production and installation of hardware as necessary.
5. **Testing and integration :**
   - This phase involves the **Quality Assurance** (QA) team who is performing the overall system testing to determine if the system solution meets the **set of business goals and if it performs as expected**.
   - All the different **components** and **subsystems** of the **solution are collect together** to bring the **whole integrated system alive**.
   - Testing is becoming important as it **helps ensure customer satisfaction** by establishing that the system is fault-free.
6. **Implementation :**
   - After the system is given **surety / Assurance** from the **Testing** or quality assurance team, it is brought into a **production environment**.
   - In essence, during this phase, the project is released to be used and/or installed by end users.

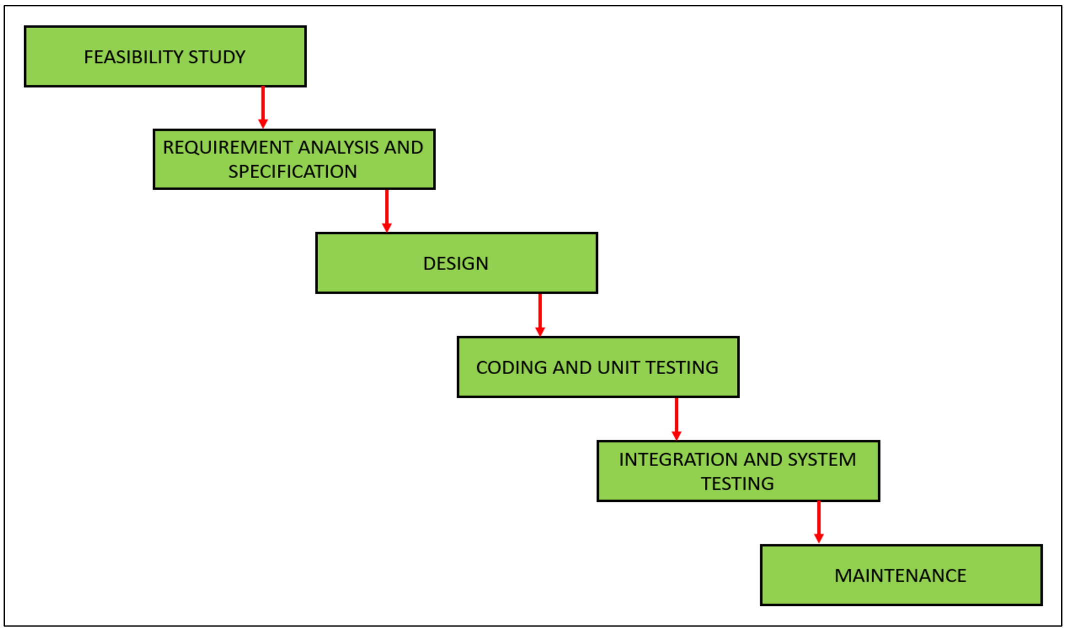
7. **Maintenance :**

- In this final phase, end users can fine-tune the system as necessary to increase performance, **add new features and capabilities**, or **meet new requirements brought to the table by the client**.
- It ensures the system remains **relevant** and usable by **replacing old hardware**, **improving the software**, **evaluating performance**, and **applying new updates** to make sure it **meets all necessary standards** and **includes the latest technologies**.

2.3 SDLC Models :

2.3.1 What is Waterfall Model :

- Classical waterfall model is the **basic software development life cycle model**.
- It is very simple but **idealistic**.
- But it is very **important** because **all the other software development life cycle models are based on the classical waterfall model**.
- Classical waterfall model divides the life cycle into a **set of phases**.
- This model considers that **one phase can be started after completion of the previous phase**.
- That is the **output** of **one phase** will be the **input to the next phase**.
- Development process can be considered as a **sequential flow** in the waterfall.
- Here the phases **do not overlap** with **each other**.


FEASIBILITY STUDY
REQUIREMENT ANALYSIS AND SPECIFICATION
DESIGN
CODING AND UNIT TESTING
INTEGRATION AND SYSTEM TESTING
MAINTENANCE

1. **Feasibility Study** :
   - ✓ This phase is to decide would be **financially** & **technically** feasible to develop **software**.
   - ✓ It involves **understanding** the **problem** and **then finds** out the **various possible strategies to solve the problem**.
   - ✓ These different **identified solutions** are **analyzed** based on their **benefits and drawbacks**, the **best solution** is **chosen** and all the other phases are carried out as per this solution strategy.
2. **Requirements analysis and specification :**
   - ✓ Requirement analysis phase is to understand the exact **requirements** of the **customer** and **document** them properly.
   - ✓ This phase consists of two different activities.
   - **Requirement Gathering and Analysis :** Firstly all the requirements regarding the **software** are **gathered** from the **customer** and then it **analyzed**. The goal of the analysis part is to ***remove incompleteness*** and **inconsistency**.
   - **Requirement Specification :** These analyzed requirements are documented in a software requirement specification (SRS) document. SRS document serves as a contract between development team and customers. Any future dispute between the customers and the developers can be settled by examining the SRS document.
3. **Design :**
   - ✓ The requirement specifications from first phase are studied in this phase and the **system design is prepared**.
   - ✓ This system design helps in specifying hardware and system requirements and helps in **defining the overall system architecture**.
4. **Coding and Unit testing :**
   - ✓ In coding phase software design is translated into **source code** using any **suitable programming language**.
   - ✓ Thus each designed module is coded.
   - ✓ The aim of the unit testing phase is to check whether each module is working properly or not.
5. **Integration and System testing :**
   - ✓ During each integration step, previously planned modules are added to the partially integrated system and the resultant system is tested.
   - ✓ Finally, after all the modules have been successfully integrated and tested, the full working system is obtained and system testing is carried out on this.
   - **Alpha testing :** This system testing performed by the **development team**.

- **Beta testing :** Beta system testing performed by **friendly set of customers**.
- **Acceptance testing :** After the **software has been delivered**, the customer performed the acceptance testing to decide whether to accept the software or to reject it.

6. **Maintenance :**

✓ Maintenance is the most important phase of a software life cycle.

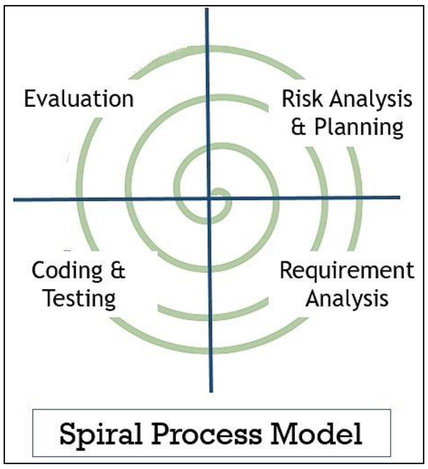
✓ There are some issues which come up in the client environment.

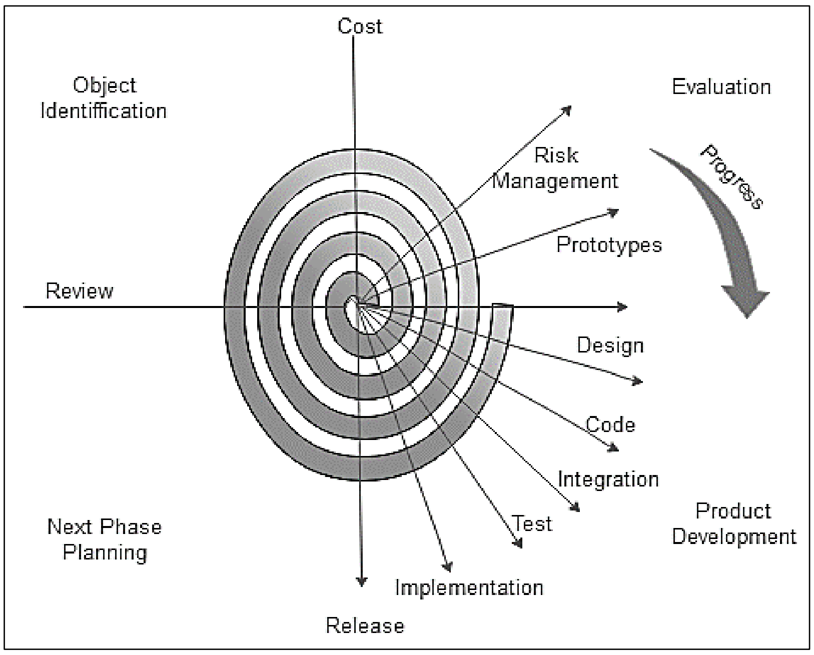
✓ There are basically three types of maintenance :

- **Corrective Maintenance :** This type of maintenance is carried out to correct errors that **were not discovered during the product development phase**.
- **Perfective Maintenance :** This type of maintenance is carried out to enhance the functionalities of the system **based on the customer's request**.
- **Adaptive Maintenance :** Adaptive maintenance is usually required for porting the software to work in a **new environment** such as work on a **new computer platform** or with a **new operating system**.

2.3.2 What is Spiral Model :

- The spiral model combines the **idea of iterative development** with the **systematic**, controlled aspects of the waterfall model.
- **Spiral model** is introduced by **Barry Boehm.**
- It decreases the uncertainty at each stage of software development.
- This model incorporates features of both the waterfall model and prototyping model.
- It is a **risk-driven process** model and its most important feature is to **decrease** the risk of the project.
- **Spiral model** is provides support for **Risk Handling**.





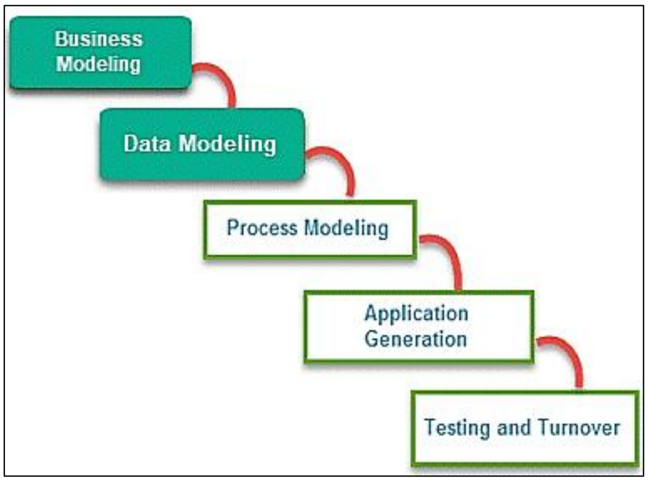
1. **Identification & Planning :**
   - ✓ This phase starts with **gathering** the business **requirements**.
   - ✓ In the successive spirals as the product matures, identification of **system requirements, subsystem requirements** & **unit requirements** are all done in this phase.
   - ✓ This phase also includes **understanding** the **system requirements** by continuous communication **between** the **customer** and the **system analyst**.
   - ✓ At the end of the spiral, the **product** is **arranged** in the identified market.
2. **Design :**
   - ✓ The Design phase starts with the **conceptual design** in the baseline spiral and involves **architectural design, logical design of modules, physical product design** and the **final design** in the successive spirals.
3. **Construct or Build :**
   - ✓ In this phase, the programs are **developed** and **integrated** to form.
   - ✓ The Construct phase refers to **production** of the **actual software product** at **every spiral**.
   - ✓ The product of this stage is tested to **find** any **error** in coding.
   - ✓ In the baseline spiral, when the product is just thought of and the **design is being developed** a POC (Proof of Concept) is developed in this phase to **get customer feedback.**

- ✓ Then in the subsequent spirals with **higher clarity** on **requirements** and **design details** a working model of the software called build is produced with a version number.
- ✓ These builds are sent to the customer for feedback.

4. **Evaluation and Risk Analysis**
   - ✓ This stage **evaluates** whether the **project is going as per planning or not**.
   - ✓ It evaluates whether the **objective set** at the **first stage has been achieved or not**.
   - ✓ The evaluation phase also helps developers to decide the number of cycles that will be required to complete the project.
   - ✓ The spiral model allows using **other process models** in one or more of its **cycle**.
   - ✓ Risk Analysis includes **identifying**, **estimating** and **monitoring** the **technical feasibility** and **Management Risks**, such as schedule slippage and cost overrun.
   - ✓ This is done to either to reduce the risk at any stage or to get the requirements of the user clear or for resolving technical risks.
   - ✓ After testing the build, at the end of first iteration, the customer evaluates the software and provides feedback.
   - ✓ The following illustration is a representation of the Spiral Model, listing the activities in each phase.

2.3.3 What is RAD Model :

- RAD stands for **Rapid Application Development Model.**
- It is a type of **incremental model**.
- In RAD model the components or functions are developed in parallel for **mini projects**.
- This can **quickly give** the **customer something** to see and use.
- It is based on iterative development with **no** specific planning or **minimal planning** involved.



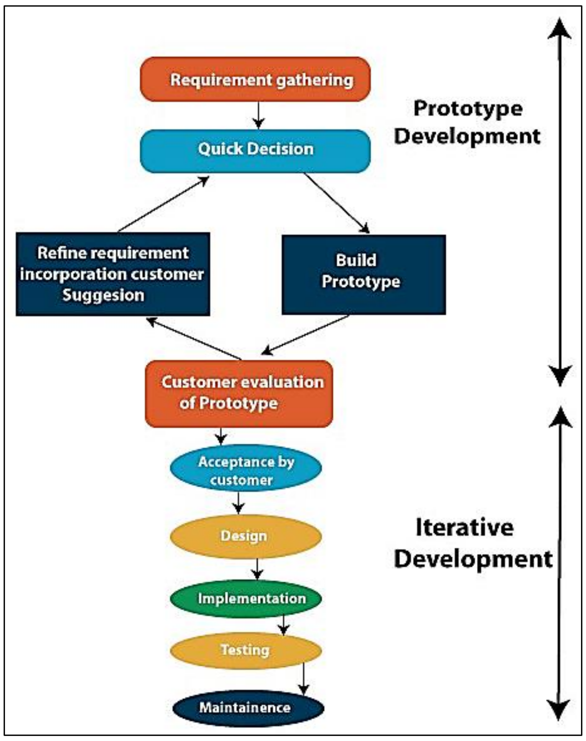
- Rapid Application Development focuses on **gathering customer requirements through focus groups, early testing of the prototypes by the customer using iterative concept, reuse of the existing components, continuous integration and rapid delivery.**
- In the RAD model, the **functional modules** are developed & make them **complete product** for **faster product delivery**.
- Since there is **no detailed preplanning**, it makes it easier to include the changes within the development process.
- RAD projects follow iterative and incremental model and have **small teams** comprising of developers, domain experts, customer representatives and other IT resources working progressively on their component or prototype.

1. **Business Modelling :**
    - The information flow is **identified between various business functions**.
    - A complete business analysis is performed to **find the vital information for business**, **how it can be obtained**, **how and when is the information processed** and **what are the factors driving successful flow of information**.
    - The business model for the product under development is designed in terms of **flow of information and the distribution of information between various business channels**.
2. **Data Modelling :**
    - Information gathered from **business modelling** is used to define data objects that are needed for the business.
    - The information gathered in the Business Modelling phase is **reviewed** and **analyzed** to form sets of data objects vital for the business.
    - The attributes of all data sets is **identified and defined**.
    - The relation between these data objects are **established and defined in detail in relevance to the business model**.
3. **Process Modelling :**
    - Data objects defined in data modeling are converted to achieve the business information flow to achieve some specific business objective.
    - Description are identified and created for CRUD of data objects.
    - The data object sets defined in the Data Modeling phase is converted to establish the business information flow needed to achieve specific business objectives as per the business model.
    - The process model for any changes or enhancements to the data object sets is defined in this phase.

- Process descriptions for **adding, deleting, retrieving or modifying a data object are given.**

4. **Application Generation :**
   - Automated tools are used to convert process models into **code and the actual system**.
   - The **actual** system is **built** and **coding** is done by using automation tools to convert process and **data models** into actual prototypes.
5. **Testing and Turnover :**
   - Test new components and all the interfaces.
   - The overall testing time is reduced in the RAD model as the prototypes are independently tested during every iteration.
   - However, the data flow and the interfaces between all the components need to be thoroughly tested with complete test coverage.
   - Since most of the programming components have already been tested, it reduces the risk of any major issues.

2.3.4 What is Prototyping Model :



Prototype Model

- Prototype is a working model of software with some **limited functionality**.
- **Prototyping Model** is a software development model in which prototype is built, tested, and reworked until an acceptable prototype is achieved.
- It also creates base to **produce the final system or software**.
- It works best in scenarios where the project's requirements are **not** known in detail.
- It is an **iterative, trial and error** method which takes place **between developer and client.**

Steps of Prototyping Model :

Step 1 : Requirements Gathering and Analysis :

- A prototyping model **starts** with **requirement** analysis.
- In this phase, the requirements of the system are **defined in detail**.
- During the process, the users of the **system** are **interviewed** to know what their **expectation** from the system is.

Step 2 : Quick Design :

- The second phase is a **preliminary design** or a **quick design**.
- In this stage, a **simple design** of the system is created.
- However, it is **not** a complete design.
- It gives a **brief idea** of the system **to the user**.
- The quick design **helps** in **developing** the **prototype**.

Step 3 : Build a Prototype :

- In this phase, an **actual prototype** is designed **based on** the **information gathered from quick design**.
- It is a **small working model** of the required system.

Step 4 : Initial User Evaluation :

- In this stage, the proposed system is presented to the **client for an initial evaluation**.
- It helps to **find out** the **strength and weakness** of the working model.
- **Comment and suggestion** are **collected from the customer** and provided to the developer.

Step 5 : Refining Prototype :

- If the user is not happy with the current prototype, you need to refine the prototype according to the **user's feedback and suggestions**.
- This phase will not over until all the requirements specified by the user are met.
- Once the **user is satisfied** with the developed prototype, a **final system is developed based on the approved final prototype.**

Step 6 : Implement Product and Maintain :

- Once the **final system is developed** based on the **final prototype**, it is thoroughly tested and deployed to production.
- The system undergoes **routine maintenance** for minimizing downtime and **prevents** large-scale failures.

Types of Prototyping Model :

1. **Rapid Prototype :**
   - Throwaway prototyping is also called as **rapid** or **close** ended prototyping.
   - Rapid throwaway is based on the **preliminary requirement**.
   - This type of prototyping uses **very little efforts** with **minimum requirement** analysis to build a prototype.
   - It is **quickly developed** to show how the requirement will look visually.
   - The **customer's feedback** helps drives changes to the requirement.
   - This technique is useful for exploring ideas and getting **instant feedback** for customer requirements.
2. **Evolutionary Prototyping :**
   - Evolutionary prototyping also called as **breadboard prototyping.**
   - It is based on building **actual functional** prototypes with **minimal functionality** in the beginning.
   - It is helpful when the requirement is **not stable** or **not understood** clearly at the **initial** stage.
   - This model is helpful for a project which uses a **new technology** that is not well understood.
   - Here, the prototype developed is incrementally refined based on **customer's feedback until** it is **finally accepted**.
   - It is also used for a **complex project**.
3. **Incremental Prototyping :**
   - The **different prototypes** are **merged into a single product**.
   - Incremental prototyping refers to building **multiple functional prototypes** of the various **sub-systems.**
   - In incremental Prototyping, the final product is destroy into different small prototypes and developed individually.
   - This method is helpful to reduce the feedback time between the user and the application development team.
4. **Extreme Prototyping :**
   - Extreme prototyping is used in the **web development domain**.

- It consists of **three sequential** phases.
  - **i.** Basic prototype with the **entire existing page** is present in the **HTML format**.
  - **ii.** You **can reproduce data process** using a prototype services layer.
  - **iii.** The **services are implemented and integrated** into the **final prototype**.

: QUESTIONS :

1. Define System Analyst.
2. Explain Role of System Analyst.
3. Explain in detail SDLC with suitable diagram.
4. What is Waterfall Model? Explain in detail.
5. What is Spiral Model? Explain in detail.
6. Explain Rapid Application Model (RAD) in detail.
7. Describe Prototype Model with suitable diagram.

***

3. Chapter

System Planning

Contents:

(10 L, 15 M)

3.1 Data and Fact Gathering Techniques :

- ✓ Many situations arise for fact-finding during the **database system development life cycle**.
- ✓ However, fact-finding is particularly vital to the **early stages** of the life cycle, which includes database planning, system definition, and requirements gathering, and analysis stages.
- ✓ It is during these early stages where the database developer captures the necessary facts essential to **build** the required database.
- ✓ Fact-finding is also used in the case of database design and the later stages of the life cycle but to a lesser extent.
- ✓ It is to be noted that it is important to make a rough estimation of how much time and effort is required to be spent on fact-finding for a database project.

1. **Interviews :**
   - 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 by which analyst learn about the existing system, its problem and expectation with the system.

- 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.
- 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.

2. **Questionnaires :**
   - This method seeks information from the person in **written** format.
   - This is a quickest way for gathering information if respondents are scattered geographically or there is no time for the interviews.
   - **Structured Question :** where the answers are in the form of YES/NO, multiple choice option selection, ratings, fill in the blanks.
   - **Unstructured Questions :** where person is asked for his opinion and he/she can answer it freely.
3. **Group Discussions :**
   - This method is often used when there no time for personal interview and information is required from **face to face** sessions.
   - As there are many person present many type of ideas can be heard.
   - Scheduling such sessions is a skillful matter because it has many problems such as : discussion may be dominated by one person others may shy to respond, presence of seniors in the group may not allow others to present their views freely, discussion may lead to verbal fight etc.
4. **Presentation :**
   - Sometime presentation can also be conducted by analyst for presenting his understanding for the system and problems with it.
   - Such presentation may include showing slide, interacting with people and talking to them **regarding system, asking questions, answering questions** etc.
   - Presentations are useful when users are passive or too busy to actively explain things.
5. **Onsite Observations :**
   - It is the process of examining the problems which had previously solved by other sources that can be either **human or documents**.
   - To solve the **requirements of problem**, the analyst ***visits to other organization*** that had previously experienced for similar problems.

- In addition, the analyst can also find the information from **database, reference books, case studies and Internet**.

6. **Observation :**
   - Another **fact finding technique** is observation.
   - In this technique, system analyst participates in the organization, studies the flow of documents, applies the existing system, and interacts with the users.
   - Observation can be a useful technique when the system analyst have user point of view.
   - Sampling technique called **work sampling** is useful for observation.
   - By using this technique, system analyst can know **how employees spend their days**.

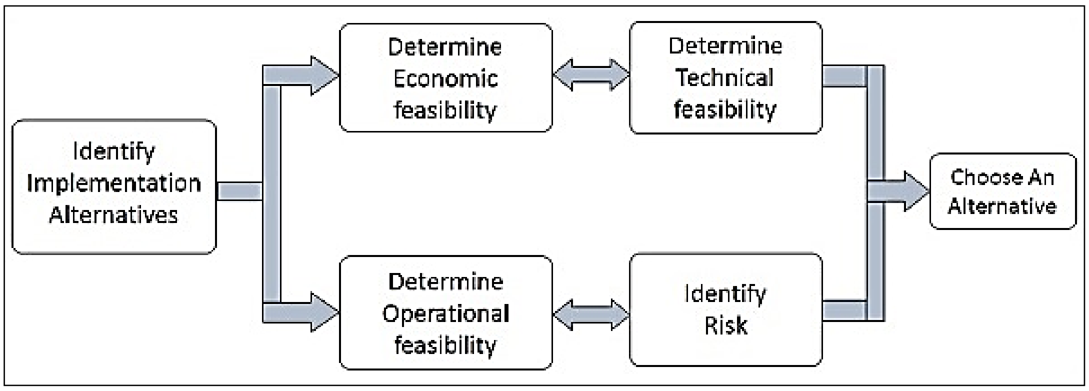
3.2 Introduction to Feasibility Study :

- Feasibility Study can be considered as **preliminary investigation.**
- It helps the **management** to take **decision** about whether study of system should be feasible for development or not.
- It identifies the **possibility of improving an existing system**, **developing a new system,** and **produce refined estimates** for ***further*** development of system.
- It is used to obtain the **outline** of the **problem** and decide whether feasible or appropriate solution exists or not.
- The main objective of a feasibility study is to acquire problem scope instead of solving the problem.
- The output of a feasibility study is a **formal system proposal act** as decision document which includes the complete nature and scope of the proposed system.

Steps Involved in Feasibility Analysis :

The following steps are to be followed while performing feasibility analysis -

1. Form a project **team** and **appoint a project leader**.
2. Develop system **flowcharts**.
3. **Identify the problems** of current system and **set goals**.
4. List the **alternative solution** to **achieve goals**.
5. Decide the feasibility of **each alternative** such as technical feasibility, operational feasibility, etc.
6. Weight the **performance** and **cost effectiveness** of each alternatives.
7. **Rank** the other alternatives and **select best** system.
8. Prepare a **system proposal** of final project directive to **management** for **approval**.


Identify Implementation Alternatives
Determine Economic feasibility
Determine Technical feasibility
Determine Operational feasibility
Identify Risk
Choose An Alternative

3.3 Types of Feasibility Study :

1. **Economic Feasibility :**
   - It is evaluating the **effectiveness** of candidate system by using **cost/benefit analysis** method.
   - It demonstrates the net benefit from the candidate system in terms of benefits and costs to the organization.
   - The main aim of Economic Feasibility Analysis (EFS) is to **estimate the economic requirements of candidate system before investments funds are committed to proposal**.
   - It prefers the alternative which will maximize the net worth of organization by earliest and highest return of funds along with lowest level of risk involved in developing the candidate system.
2. **Technical Feasibility :**
   - It investigates the technical feasibility of **each implementation alternative**.
   - It analyzes and **determines** whether the **solution can be supported by existing technology or not.**
   - The analyst determines whether **current technical resources** be upgraded or **added it that fulfil the new requirements**.
   - It ensures that the **candidate system provides appropriate responses** to **what extent it can support the technical enhancement**.
3. **Operational Feasibility :**
   - It determines whether the system is **operating effectively** once it is developed and implemented.
   - It ensures that the **management** should support the proposed system and its working feasible in the **current organizational environment**.

- It analyzes whether the users will be affected and they **accept the modified** or **new business methods that affect the possible system benefits**.
- It also **ensures** that the **computer resources** and **network architecture of candidate system are workable**.

4. **Behavioural Feasibility :**
   - It **evaluates** and **estimates** the **user attitude** or **behavior** towards the **development of new system**.
   - It helps in determining if the system requires special effort to **educate, retrain, transfer, and changes** in **employee's job status on new ways** of conducting business.
5. **Schedule Feasibility :**
   - It ensures that the project should be **completed** within given **time constraint** or **schedule**.
   - It also **verifies** and **validates** whether the ***deadlines*** of project are reasonable or not.

3.4 Introduction to Software Requirement Specifications (SRS) :

- A software requirements specification (SRS) is a document that captures complete description about how the system is expected to perform.
- A **software requirements specification** (**SRS**) is a description of a software system to be developed.
- The SRS fully describes **what the software will do** and **how it will be expected** to perform.
- An SRS **minimizes** the **time** and effort required by developers to **achieve desired goals** and also **minimizes the development cost**.
- A good SRS defines **how an application will interact with system hardware**, other programs and **human users** in a wide variety of real-world situations.
- Parameters such as operating **speed, response time, availability, portability, maintainability, footprint, security and speed of recovery** from adverse events are evaluated.
- It should also provide a **realistic** basis for **estimating product costs, risks, and schedules**.
- Software requirements specifications establish the basis for an **agreement between customers** on **how the software product should function.**
- Software requirements specification is a difficult assessment of requirements before the more specific system design stages, and its goal is to reduce later redesign.

3.5 Need of Software Requirement Specifications (SRS) :

1. **Correctness :** The **document** should **reflect** the **correct specifications** and **functionalities** of the product at any **given time**.
2. **Unambiguousness :** An SRS document is the core of a software development project. Thus, it needs to be **clear** and **on point**. There should not be any vague or unclear meanings and details. Moreover, having a comprehensive glossary at the end with a **proper explanation** can be highly beneficial.
3. **Consistency :** The SRS document should align with other **technical** and **functional documents** like system requirements. It should be consistent throughout, **without changing** any information within itself.
4. **Completeness :** It should have all the **necessary information and requirements** that might be helpful during and **after the development phase**.
5. **Verifiability :** Every statement present in the document should be verifiable. **It will help ensure that the system is meeting the requirements.** Having ambiguous statements and expectations that cannot be verified will lead to complications.
6. **Ranking requirements per importance/stability :** An organization's software requirements are not equally critical. While some might need immediate attention, others can be taken up based on priority. **The SRS document helps in highlighting the ranking of the software features in terms of importance and stability**.
7. **Modifiability :** The software development process is continuous and might require some **changes** as the development begins. Therefore, the SRS should be **flexible** to enable **owners to modify it easily per the requirement**.
8. **Traceability :** Every requirement should have a clear origin with references. It will **help** in making the **feature's lifecycle traceable**.

: QUESTIONS :

1. Explain different Data and Fact Gathering Techniques.
2. What is Feasibility Study? Explain in detail.
3. Explain different Types of Feasibility Study.
4. What is SRS? Explain its needs in software engineering.

***

4. Chapter

Systems Design and Modelling

Contents :

(10 L, 15 M)

4.1 Computer Aided Software Engineering (CASE) :

4.1.1 Systems Flowcharts :

A flowchart is often used to manage, analyse, design a process in many different fields. It is a useful tool that everyone should learn to help solve problems easier and more efficiently.

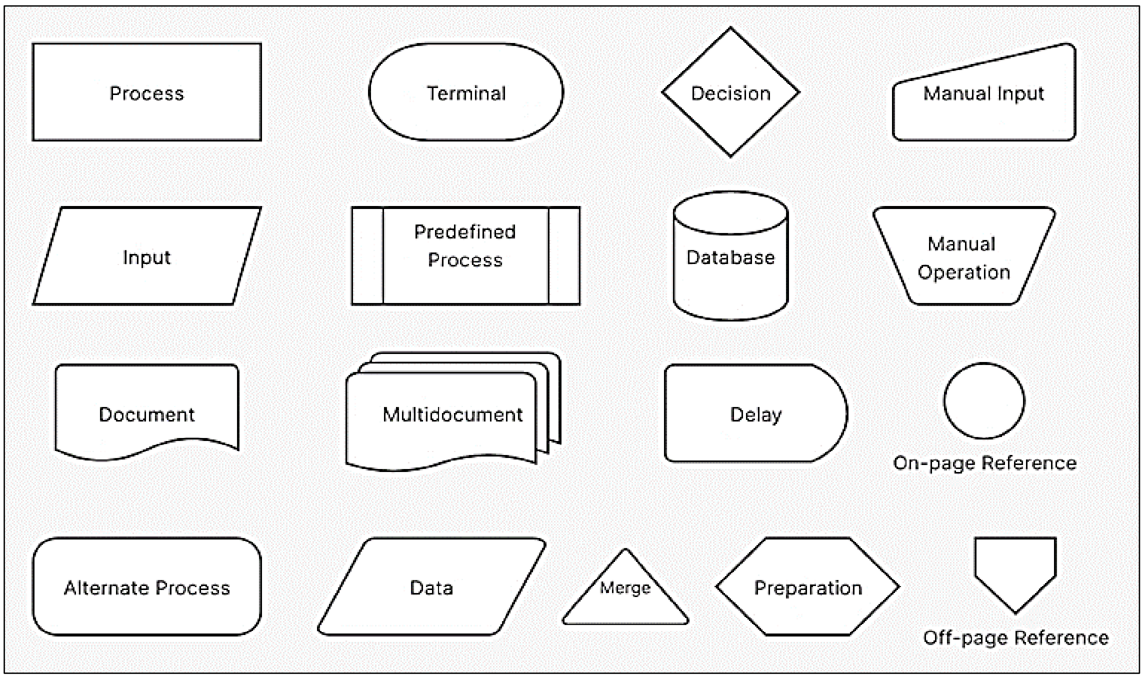
What is System Flowchart :

As you may have already known, a flow chart is a graph that shows process flow, decisions, and outcomes. They are common tools of quality control that are utilized in many fields. There are four basic categories of flowcharts :

- Document flowcharts show you the flow of documents from one business unit to another.
- Data flowcharts let you see the overall data flow in a system.
- Program flowcharts show you a program's control in a system. They are also one of the essential tools in programming. We have already covered it in an article, so check it out on our website!
- System flowcharts are the diagram type that shows you the flow of data and how decisions can affect the events surrounding it.

Like other types of flowcharts, system flowcharts consist of start/end terminals, processes, and decisions, all connected by arrows showing the flow and how data moves in the flow.

Flowchart Symbols :

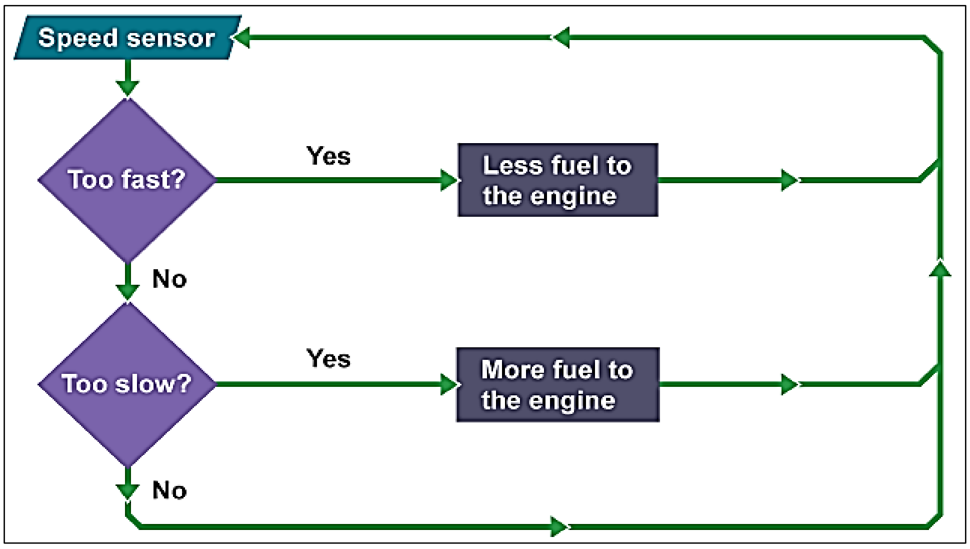


Example : A Car's Cruise Control :

In this example, we will look at a car's cruise control. Cruise control allows the car to stay at the desired speed that the driver set. It works by adding or cutting the fuel according to the car’s speed.

To keep the car at a constant speed, we will have a major component called a speed sensor that records the car's speed. If the car is too slow, we can speed it up by adding more fuel, while if the car is too fast, we can slow it down by cutting some fuel.

A simple flow chart for this system should be like this:



Benefits of System Flowchart :

- Easy understanding of system processes.
- Identifies potential bottlenecks or areas for improvement.
- Aids in system documentation and communication.

4.1.2 Data Flow Diagrams :

A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. A neat and clear DFD can depict the right amount of the system requirement graphically. It can be manual, automated, or a combination of both.

It shows how data enters and leaves the system, what changes the information, and where data is stored.

The objective of a DFD is to show the scope and boundaries of a system as a whole. It may be used as a communication tool between a system analyst and any person who plays a part in the order that acts as a starting point for redesigning a system. The DFD is also called as a data flow graph or bubble chart.

The following observations about DFDs are essential:

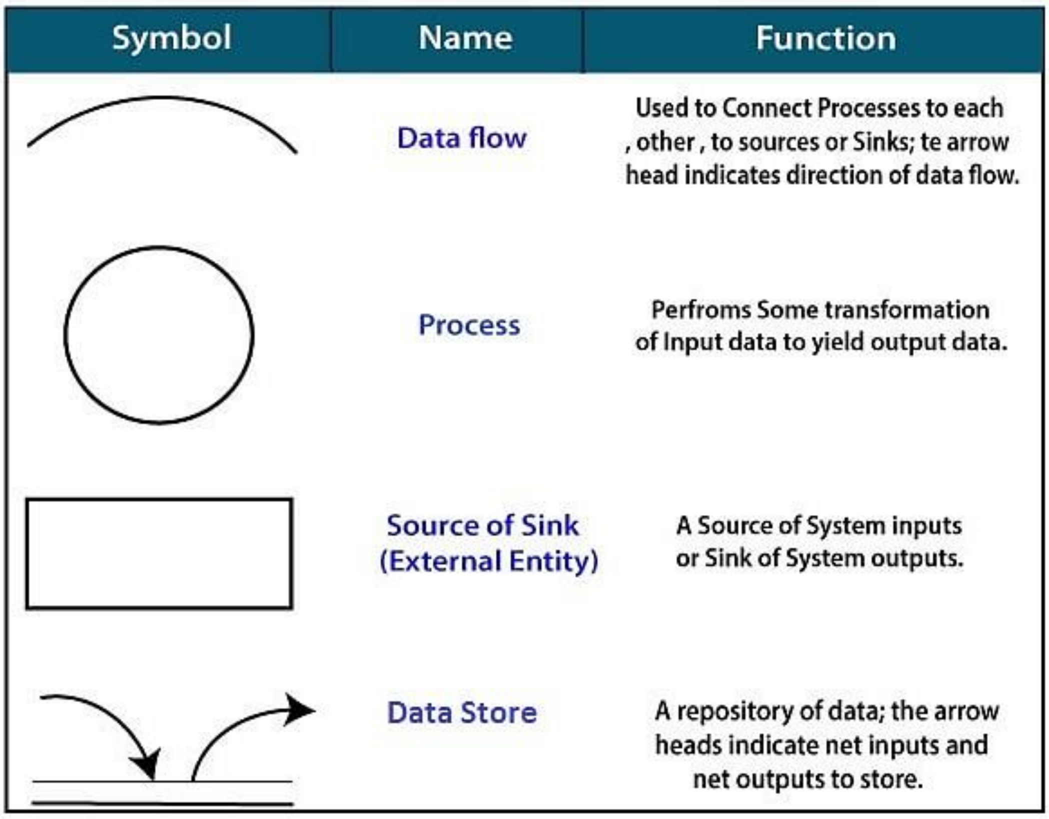
All names should be unique. This makes it easier to refer to elements in the DFD.

Remember that DFD is not a flow chart. Arrows is a flow chart that represents the order of events; arrows in DFD represents flowing data. A DFD does not involve any order of events.

Suppress logical decisions. If we ever have the urge to draw a diamond-shaped box in a DFD, suppress that urge! A diamond-shaped box is used in flow charts to represents decision points with multiple exists paths of which the only one is taken. This implies an ordering of events, which makes no sense in a DFD.

Do not become bogged down with details. Defer error conditions and error handling until the end of the analysis.

Standard symbols for DFDs are derived from the electric circuit diagram analysis and are shown in fig :



Symbol	Name	Function
	Data flow	Used to Connect Processes to each , other , to sources or Sinks; te arrow head indicates direction of data flow.
	Process	Perfroms Some transformation of Input data to yield output data.
	Source of Sink (External Entity)	A Source of System inputs or Sink of System outputs.
	Data Store	A repository of data; the arrow heads indicate net inputs and net outputs to store.

Symbols for Data Flow Diagrams

Circle : A circle (bubble) shows a process that transforms data inputs into data outputs.

Data Flow : A curved line shows the flow of data into or out of a process or data store.

Data Store : A set of parallel lines shows a place for the collection of data items. A data store indicates that the data is stored which can be used at a later stage or by the other processes in a different order. The data store can have an element or group of elements.

Source or Sink : Source or Sink is an external entity and acts as a source of system inputs or sink of system outputs.

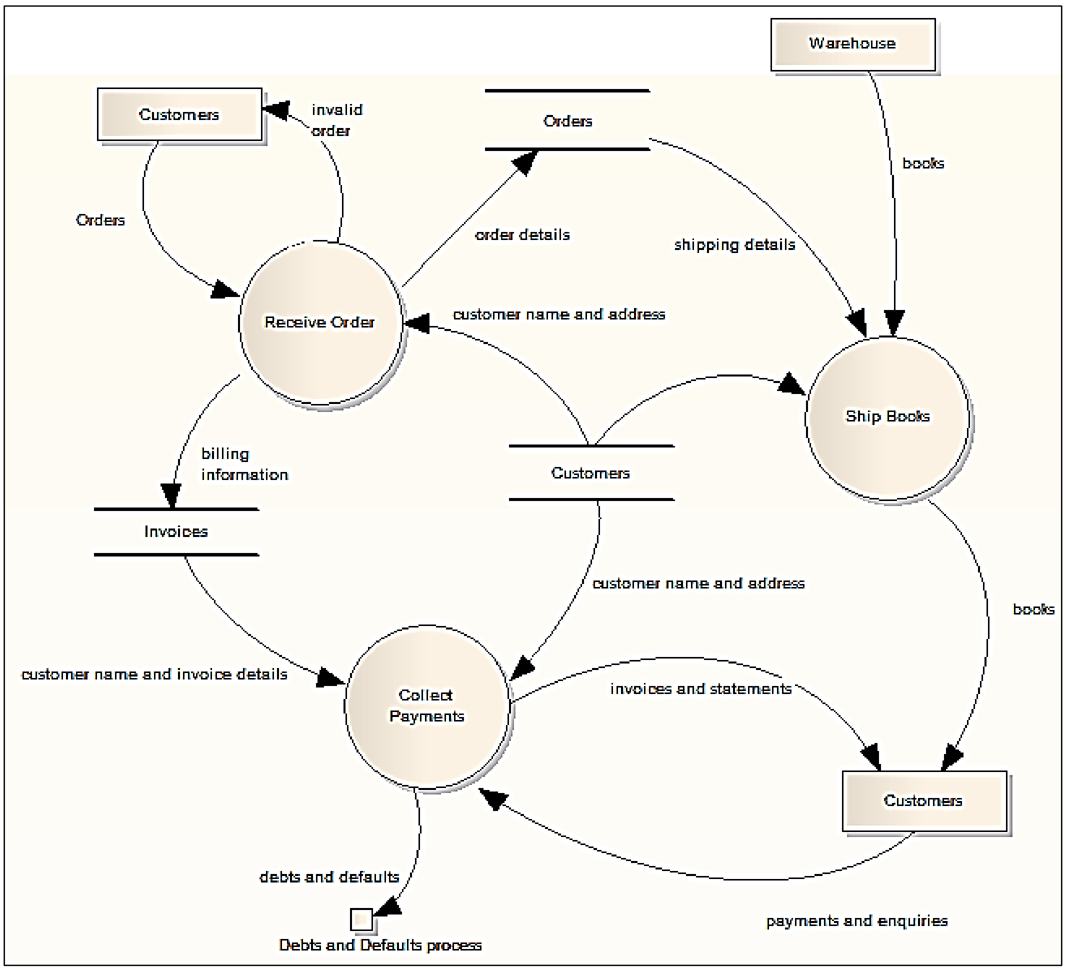
Context level Diagram :

A Context level diagram is a top-level Data Flow diagram that has just one Process element representing the system being modelled, showing its relationship to external systems.

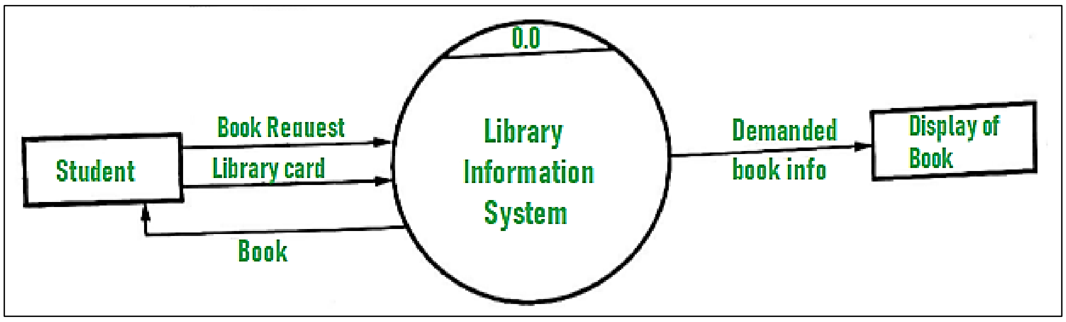
Levels in DFD :

- On next level each process is expanded to more sub process.
- Process is numbered so that it depicts or show that it is expanded from existing process.
- We can extend from level i to level (i+1), and value of "i" is depend on size and complexity of problem.
- At every next level we expand process or divide process.

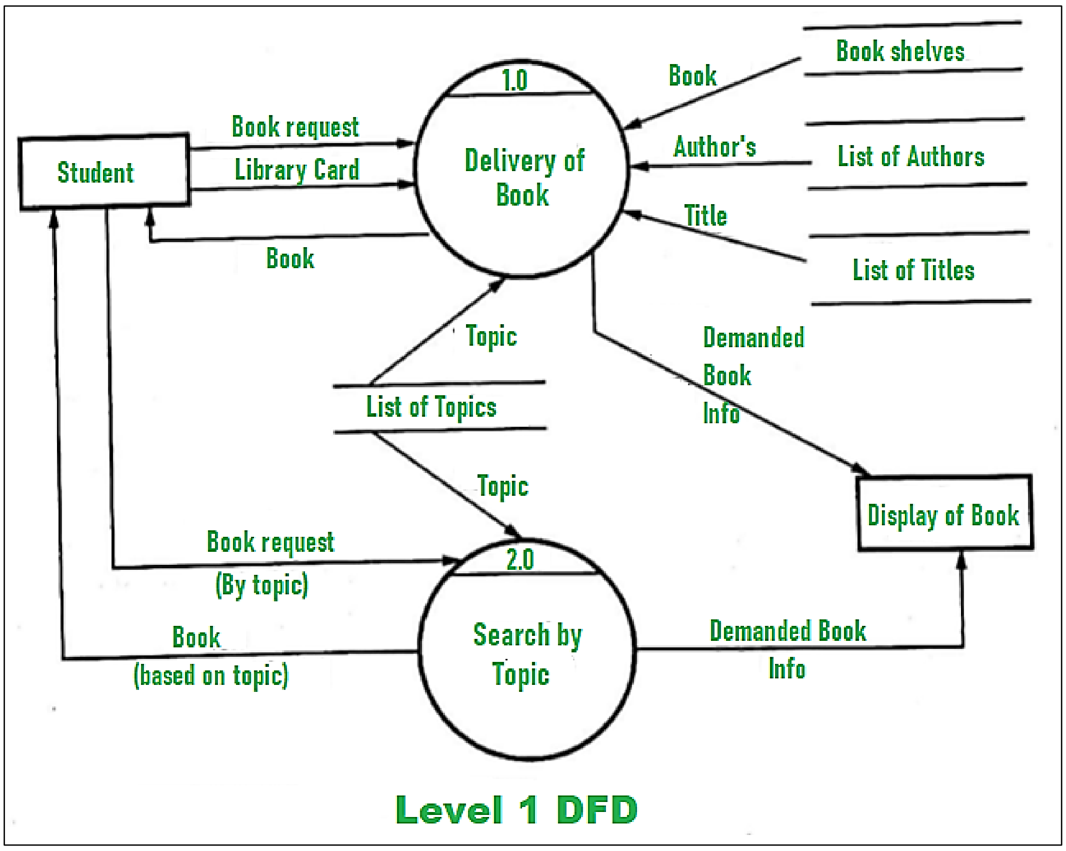
Example of Data flow diagram (book selling)

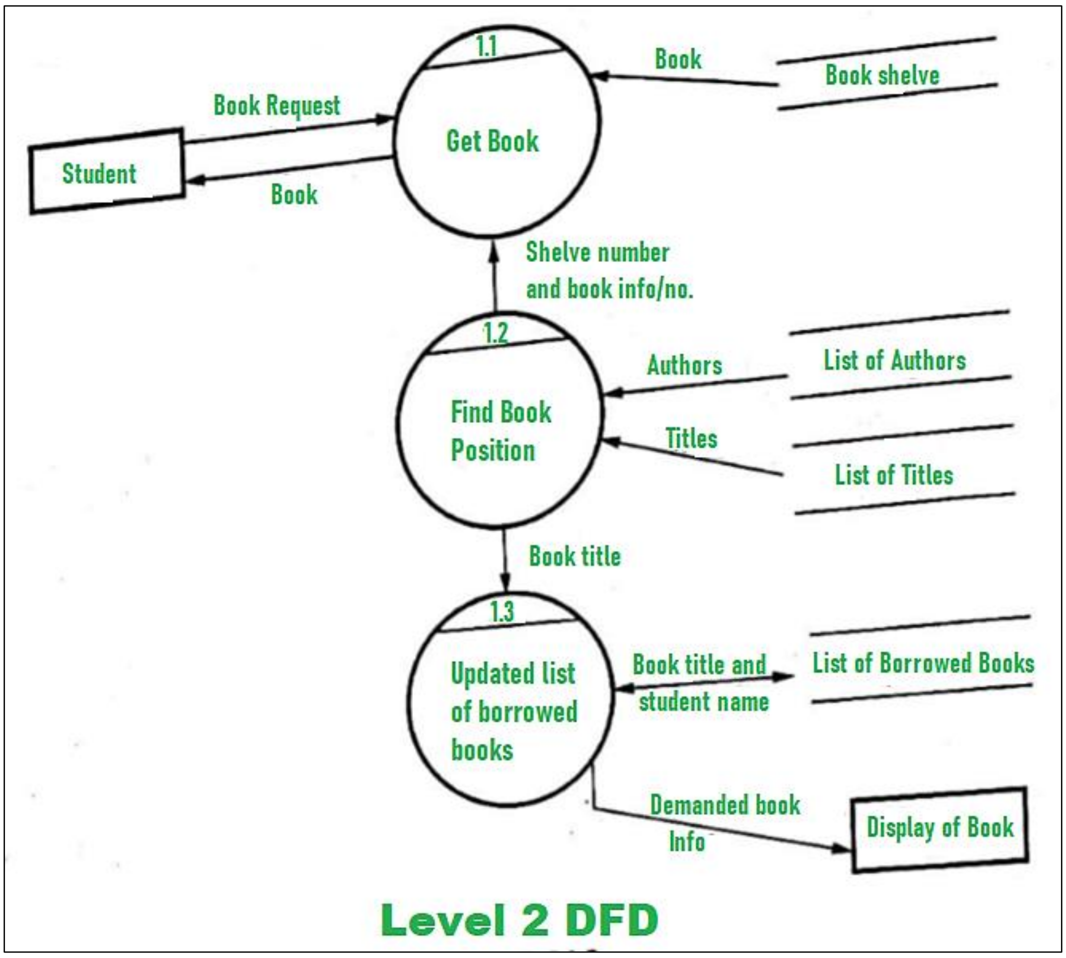


Example of DFD level 0, Level 1 and Level 2 :



Context Level DFD or Level 0 DFD





4.1.3 Entity Relationship Diagram :

ER-modeling is a data modeling method used in software engineering to produce a conceptual data model of an information system. Diagrams created using this ER-modeling method are called Entity-Relationship Diagrams or ER diagrams or ERDs.

Purpose of ERD :

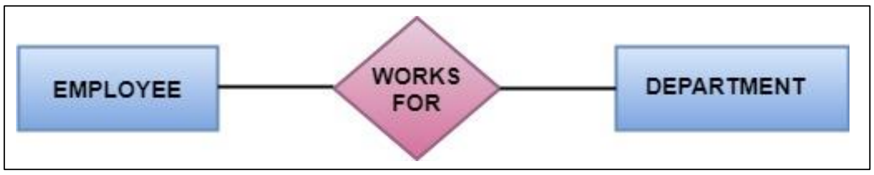
The database analyst gains a better understanding of the data to be contained in the database through the step of constructing the ERD.

The ERD serves as a documentation tool.

Finally, the ERD is used to connect the logical structure of the database to users. In particular, the ERD effectively communicates the logic of the database to users.

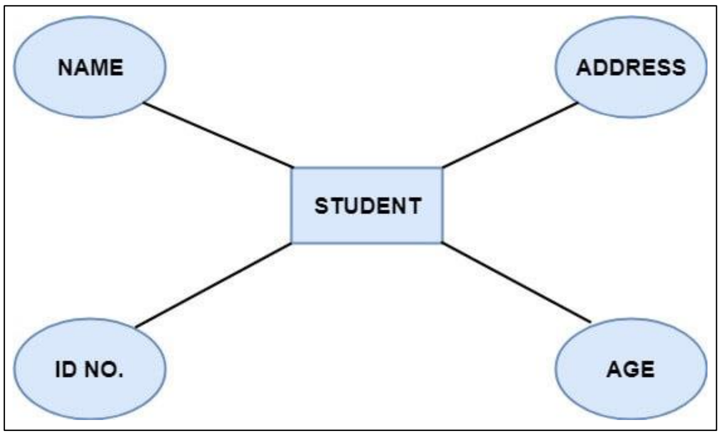
1. **Entity** : An entity can be a real-world object, either animate or inanimate, that can be merely identifiable. An entity is denoted as a rectangle in an ER diagram. For example,

in a school database, students, teachers, classes, and courses offered can be treated as entities. All these entities have some attributes or properties that give them their identity.
Entity Set : An entity set is a collection of related types of entities. An entity set may include entities with attribute sharing similar values. For example, a Student set may contain all the students of a school; likewise, a Teacher set may include all the teachers of a school from all faculties. Entity set need not be disjoint.



2. **Attributes** : Entities are denoted utilizing their properties, known as attributes. All attributes have values. For example, a student entity may have name, class, and age as attributes.

   There exists a domain or range of values that can be assigned to attributes. For example, a student's name cannot be a numeric value. It has to be alphabetic. A student's age cannot be negative, etc.



There are four types of Attributes :

1. Key attribute
2. Composite attribute
3. Single-valued attribute
4. Multi-valued attribute
5. Derived attribute

3. **Relationships** : The association among entities is known as relationship. Relationships are represented by the diamond-shaped box. For example, an employee works_at a department, a student enrolls in a course. Here, Works_at and Enrolls are called relationships.

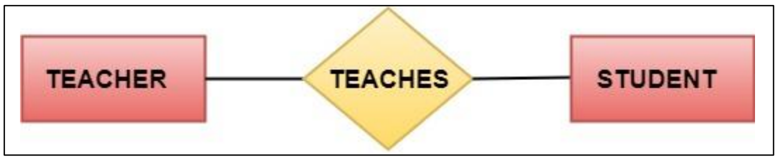


Fig. Relationships in ERD.

Relationship Set : A set of relationships of a similar type is known as a relationship set. Like entities, a relationship too can have attributes. These attributes are called descriptive attributes.

Example (Library management system) :

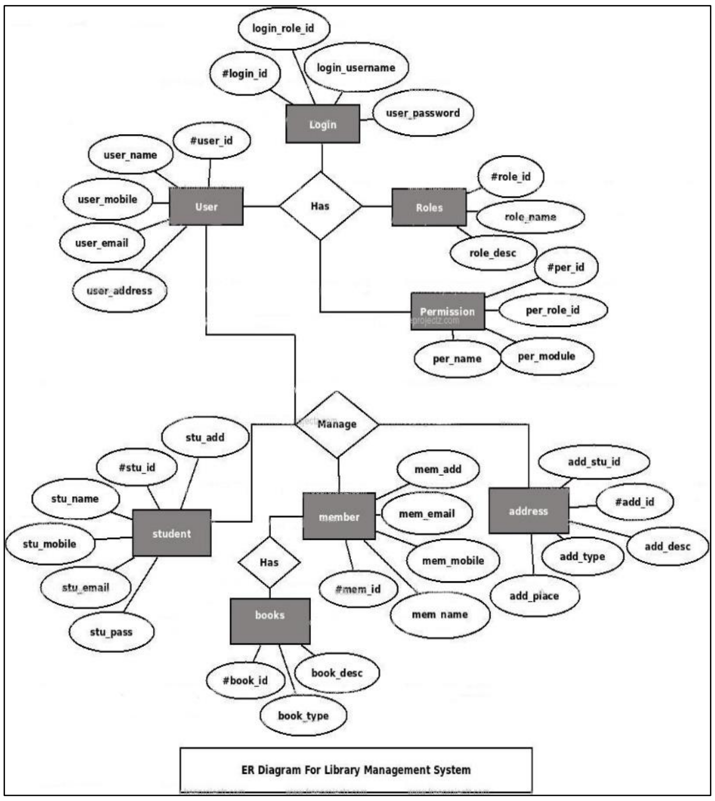
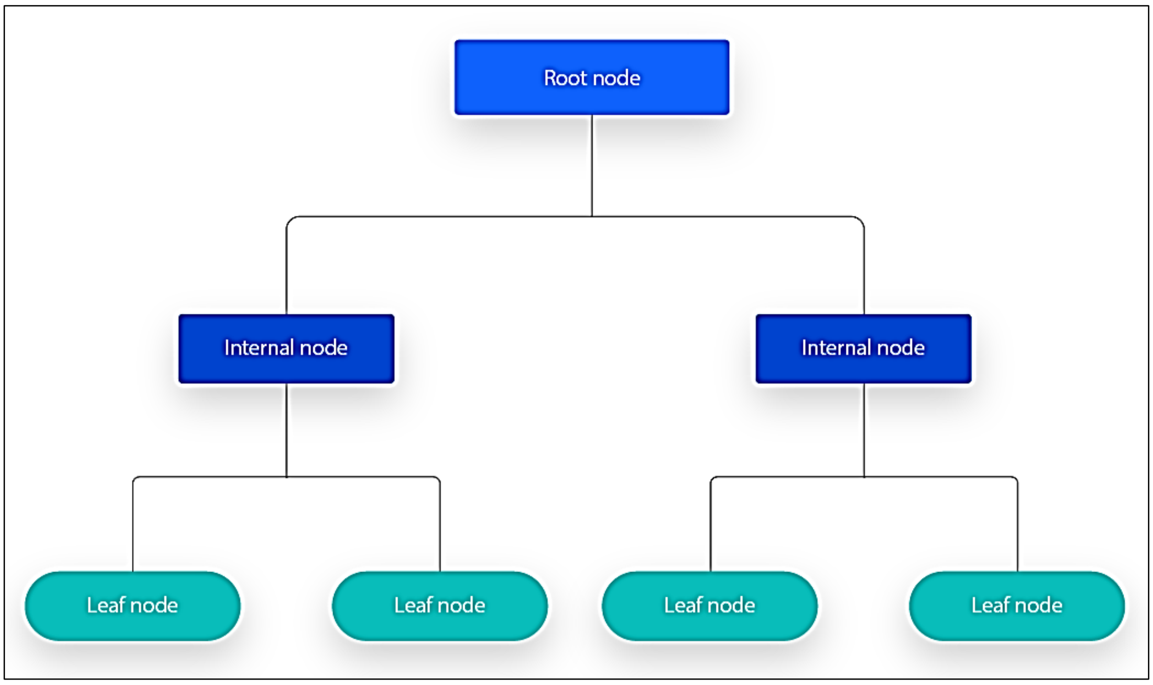


Fig. ER Diagram for Library Management System.

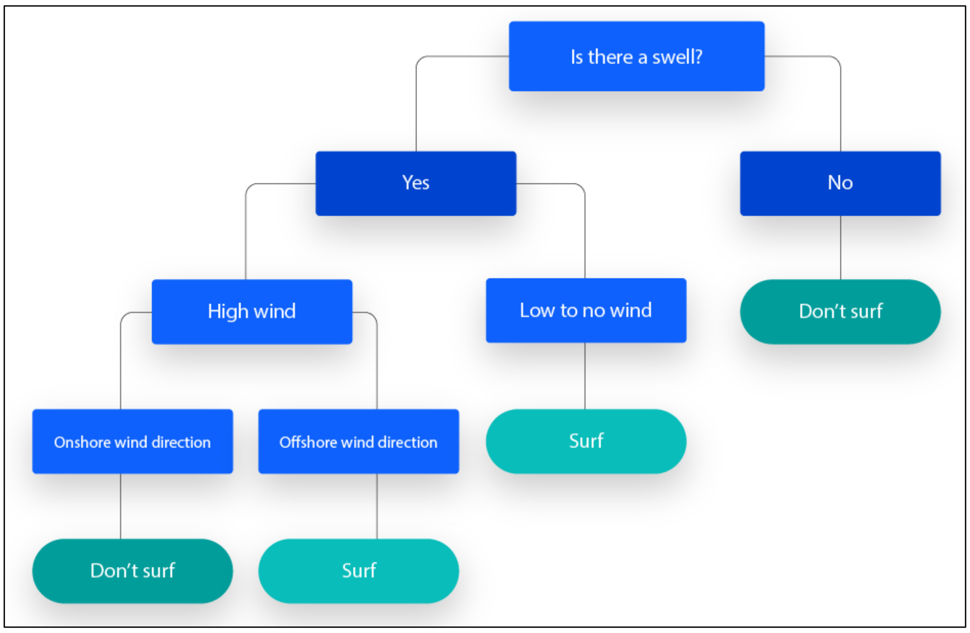
4.2 Tools for Structured Analysis :

4.2.1 Decision Tree :

A decision tree is a non-parametric supervised learning algorithm, which is utilized for both classification and regression tasks. It has a hierarchical, tree structure, which consists of a root node, branches, internal nodes and leaf nodes.



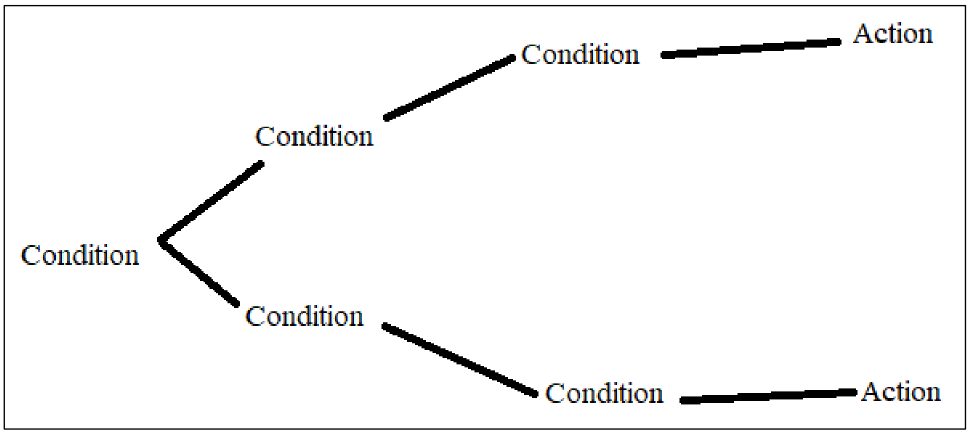
As you can see from the diagram above, a decision tree starts with a root node, which does not have any incoming branches. The outgoing branches from the root node then feed into the internal nodes, also known as decision nodes. Based on the available features, both node types conduct evaluations to form homogenous subsets, which are denoted by leaf nodes, or terminal nodes. The leaf nodes represent all the possible outcomes within the dataset. As an example, let’s imagine that you were trying to assess whether or not you should go surf, you may use the following decision rules to make a choice:



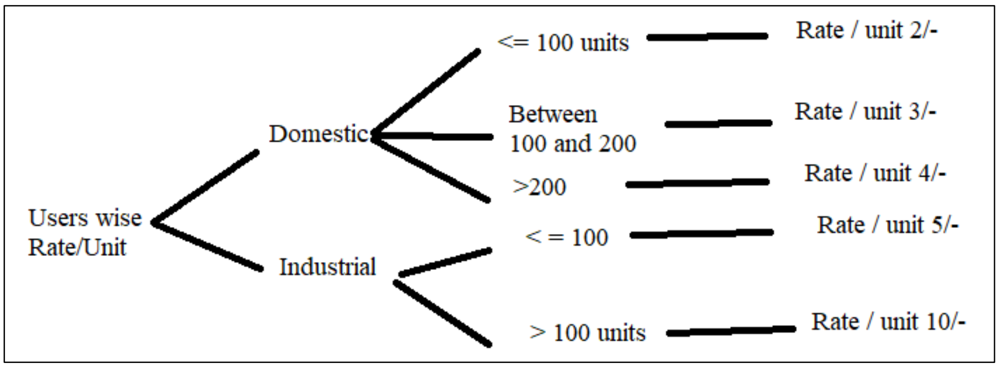
This type of flowchart structure also creates an easy to digest representation of decision-making, allowing different groups across an organization to better understand why a decision was made.

Decision tree learning employs a divide and conquer strategy by conducting a greedy search to identify the optimal split points within a tree. This process of splitting is then repeated in a top-down, recursive manner until all, or the majority of records have been classified under specific class labels. Whether or not all data points are classified as homogenous sets is largely dependent on the complexity of the decision tree. Smaller trees are more easily able to attain pure leaf nodes—i.e. data points in a single class. However, as a tree grows in size, it becomes increasingly difficult to maintain this purity, and it usually results in too little data falling within a given subtree. When this occurs, it is known as data fragmentation, and it can often lead to overfitting. As a result, decision trees have preference for small trees, which is consistent with the principle of parsimony in Occam's Razor; that is, "entities should not be multiplied beyond necessity." Said differently, decision trees should add complexity only if necessary, as the simplest explanation is often the best. To reduce complexity and prevent overfitting, pruning is usually employed; this is a process, which removes branches that split on features with low importance. The model's fit can then be evaluated through the process of cross-validation. Another way that decision trees can maintain their accuracy is by forming an ensemble via a random forest algorithm; this classifier predicts more accurate results, particularly when the individual trees are uncorrelated with each other.

Normally decision tree is shown below :



Consider example of rate per units for electric unites consumption for domestic and industrial users.



4.2.2 Decision Tables :

Decision Table Testing is a software testing methodology used to test system behaviour for various input combinations. In this systematic approach, the several input combinations and their corresponding system behaviour are represented in tabular form. The decision table is also called a Cause-Effect table, as the causes and effects for comprehensive test coverage are captured in this table. Decision Table testing is a commonly used black-box testing technique and is ideal for testing two or more inputs that have a logical relationship.

What is a Decision Table?

A decision table is the tabular representation of several input values, cases, rules, and test conditions. The Decision table is a highly effective tool utilized for both requirements management and complex software testing. Through this table, we can check and verify all possible combinations of testing conditions. The testers can quickly identify any skipped needs by reviewing the True(T) and False(F) values assigned for these conditions.

Structure of decision table is as shown bellow :

CONDITIONS Action1 Action2 Action3 Action4 …

Condition 1

Condition 2

Condition 3

Condition 4

.

.

Consider example of rate per units for electric unites consumption for domestic and industrial users.

Condition / Action	**Rate / unit Rs. 2/-**	**Rate / unit Rs. 3/-**	**Rate / unit Rs. 4/-**	**Rate / unit Rs. 5/-**	**Rate / unit Rs. 10/-**	**Rate / unit Rs. 15/-**
< 100 unis	Y			Y		
Between 100 and 200		Y			Y	
Above 200			Y			Y
Domestic	Y	Y	Y			
Industrial				Y	Y	Y

Advantages of Decision Table Testing :

1. Decision tables are one of the most effective and full-proof design testing techniques.
2. Testers can use decision table testing to test the results of several input combinations and software states.
3. It gives the developers to state and analyzes complex business rules.
4. Decision table testing is the most preferred black box testing and requirements management.
5. A decision table is used for modelling complex business logic. They can first be converted to test cases and test scenarios through decision table testing.
6. This technique provides comprehensive coverage of all test cases that can significantly reduce the re-work on writing test cases and test scenarios.
7. Decision tables guarantee coverage of all possible combinations of condition values which are called completeness property.
8. Decision tables can be used iteratively. The table results created in the first testing iteration can be used for the next and so on.
9. Decision tables are easy to understand, and everyone can use and implement this design and testing method, scenarios and test cases without prior experience.
10. Multiple conditions, scenarios and results can be viewed and analyzed on the same page by both developers and testers.

4.2.3 Structured English :

Structured English uses a simple English language to make structured programming easier for non-professionals. The program is divided into various parts that are known as logical statements. These statements are written in a simple English language.

Structured English is based on the structural logic to create a narrative form of English. It uses a series of blocks that are used for indentation, and it uses capitalization to represent the structure of logical specification hierarchically. This method states the rules that are used, and it does not show any other rules or decisions. All the rules are used for an organization or an individual in the situation, where they try to overcome the issues faced because of an ambiguous language. These problems occur because of the actions and conditions that are used for formatting procedures and decisions. Structured English uses iterations, formulas, or case words to represent decision making. Sequence structures, decision structures, case structures, and iterations are used to express the logical terms in structured English. These can be considered as subsets of English vocabulary that are used to express the procedures that are performed.

Elements of Structured English :

Structured English can be considered a pseudo-code, in a limited form. It is made up of various elements. The order in which the statement is executed is from the first line of code to the last line. Language is mainly made of Conditional blocks and repetition blocks. Keywords such as, IF, END IF, THEN, and ELSE are used to indicate the conditional statements. Additionally, the keywords such as DO, WHILE, and UNTIL are used to indicate repetition statements.

: QUESTIONS :

1. Write short note on system flowchart.
2. What is Data Flow Diagram (DFD), What are different symbols of DFD? What are advantages of DFD?
3. Draw DFD for Library system up to level 2.
4. Draw DFD for Admission system up to level 2.
5. What is purpose of ER diagram? State different symbols of DFD.
6. What is ER diagram? State advantages of ER diagram.
7. Draw ER diagram for Library system of a college.
8. What is purpose of Decision tree? Draw structure of Decision tree.
9. Draw Decision tree for fix deposit scheme of a bank.
10. What is purpose of Decision table? Draw structure of Decision table.
11. Draw Decision table for fix deposit scheme of a bank.
12. Differentiate decision tree and decision table.
13. Write short note on structure English.

***

5. Chapter

User Interface of System & Software Testing

C o n t e n t s :

(10 L, 15 M)

5.1 User - Interface Design :

The visual part of a computer application or operating system through which a client interacts with a computer or software. It determines how commands are given to the computer or the program and how data is displayed on the screen.

Types of User Interface :

There are two main types of User Interface :

- Text-Based User Interface or Command Line Interface.
- Graphical User Interface (GUI).

Text-Based User Interface : This method relies primarily on the keyboard. A typical example of this is UNIX.

Advantages :

- Many and easier to customizations options.
- Typically capable of more important tasks.

Disadvantages :

- Relies heavily on recall rather than recognition.
- Navigation is often more difficult.

- **Graphical User Interface (GUI) :** GUI relies much more heavily on the mouse. A typical example of this type of interface is any versions of the Windows operating systems.
- GUI Characteristics

Characteristics	**Descriptions**
Windows	Multiple windows allow different information to be displayed simultaneously on the user's screen.
Icons	Icons different types of information. On some systems, icons represent files. On other icons describes processes.
Menus	Commands are selected from a menu rather than type in a command language.
Pointing	A pointing device such as a mouse is used for selecting choices from a menu or indicating items of interests in a windows.
Graphics	Graphics elements can be mixed with text or the same display.

Advantages :

- Less expert knowledge is required to use it.
- Easier to Navigate and can look through folders quickly in a guess and check manner.
- The user may switch quickly from one task to another and can interact with several different applications.

Disadvantages :

- Typically decreased options.
- Usually less customizable. Not easy to use one button for tons of different variations.

5.2 Graphical Interfaces :

A GUI is the interface with which a user engages when interacting with electronic devices. We can categorize a GUI's components into three different categories.

Input Controls: We use input controls to get information from the user on what tasks they want to perform. Input controls include buttons, text fields, checkboxes, dropdown lists and list boxes.

Navigational Components: Navigational components include items that control the movement from one GUI to another. For example if you're on LinkedIn, you can click on "My Feed" to see the posts your network shared or "My Profile" to view your profile. These pieces of clickable text are examples of navigational components because they allow you to navigate through the different web pages of the website. Navigational components can also include items like sliders and search fields.

Informational Components: Informational components are those that deliver a piece of information for the user about the status of a task or other system information. For example, a progress bar, notification icon or message box are all considered informational components.

Advantages of Graphical User Interfaces :

GUIs have transformed how we view and interact with technology and they have many advantages.

- GUIs are simple to use and understand.
- GUIs are convenient.
- GUIs don't require prior computer knowledge to operate.
- GUIs allow for multitasking operations on the system.
- GUIs have instant results. In other words, you move to the next page or task right away when you click a button rather than writing lines of commands to achieve the same outcome.

What Makes a Good GUI?

Since almost everyone interacts with GUIs as a part of their daily lives, an important question to ask is: What makes a good, effective GUI? Generally, there are a few design choices that make a good GUI.

Simplicity : The simpler the design, the easier it will be for the users to handle and adapt the GUI for daily use.

Consistent use of elements : When we design a website, for example, it's important to keep the colors and overall theme consistent throughout the website, which makes the platform easier to navigate.

Incorporating color theory : The choice of colors can tremendously change how the user perceives an icon or an element on the screen, so making the correct choice is critical. For example, we often use the color green to mean "proceed," while red is used to convey "stop." So, if I build a GUI and make the OK button red and the cancel green, it may confuse users.

Clear transitions between the different parts of the GUI : For example, if we consider Google search, the transition from the page where you type your query to the results page is subtle and fast. The user can intuit how to move between pages. The buttons are strategically placed and movement between the pages is smooth.

The design choices that can make the difference between a good GUI and a bad one are the job of the UX designer. Ultimately, what makes a good GUI can differ significantly based on the purpose of the application and the target audience.

5.3 Elements of Good Interface Design :

Simplicity :

User Interface design should be simple.

Less number of mouse clicks and keystrokes are required to accomplish this task.

It is important that new features only add if there is compelling need for them and they add significant values to the application.

Consistency :

The user interface should have a more consistency.

Consistency also prevents online designers information chaos, ambiguity and instability.

We should apply typeface, style and size convention in a consistent manner to all screen components that will add screen learning and improve screen readability. In this we can provide permanent objects as unchanging reference points around which the user can navigate.

Intuitiveness :

The most important quality of good user interface design is intuitive.

Intuitive user interface design is one that is easy to learn so that user can pick it up quickly and easily.

Icons and labels should be concise and cogent. A clear unambiguous icon can help to make user interface intuitive and a good practice is make labels conform to the terminology that the application supports.

Prevention :

A good user interface design should prevents users from performing an in-appropriate task and this is accomplished by disabling or “graying cut” certain elements under certain conditions.

Forgiveness :

This quality can encourage users to use the software in a full extent.

Designers should provide users with a way out when users find themselves somewhere they should not go.

Graphical User Interface Design :

A graphic user interface design provides screen displays that create an operating environment for the user and form an explicit visual and functional context for user’s actions.

It includes standard objects like buttons, icons, text, field, windows, images, pull-down and pop-up screen menus.

5.4 Introduction to Software Testing :

Software testing can be stated as the process of verifying and validating whether a software or application is bug-free, meets the technical requirements as guided by its design and development, and meets the user requirements effectively and efficiently by handling all the exceptional and boundary cases. The process of software testing aims not only at finding faults in the existing software but also at finding measures to improve the software in terms of efficiency, accuracy, and usability. The article focuses on discussing Software Testing in detail.

What is Software Testing?

Software Testing is a method to assess the functionality of the software program. The process checks whether the actual software matches the expected requirements and ensures the software is bug-free. The purpose of software testing is to identify the errors, faults, or missing requirements in contrast to actual requirements. It mainly aims at measuring the specification, functionality, and performance of a software program or application.

Software testing can be divided into two steps :

Verification : It refers to the set of tasks that ensure that the software correctly implements a specific function. It means "Are we building the product right?".

Validation : It refers to a different set of tasks that ensure that the software that has been built is traceable to customer requirements. It means "Are we building the right product?".

Importance of Software Testing :

Defects can be identified early : Software testing is important because if there are any bugs they can be identified early and can be fixed before the delivery of the software.

Improves quality of software : Software Testing uncovers the defects in the software, and fixing them improves the quality of the software.

Increased customer satisfaction : Software testing ensures reliability, security, and high performance which results in saving time, costs, and customer satisfaction.

Helps with scalability : Software testing type non-functional testing helps to identify the scalability issues and the point where an application might stop working.

Saves time and money : After the application is launched it will be very difficult to trace and resolve the issues, as performing this activity will incur more costs and time. Thus, it is better to conduct software testing at regular intervals during software development.

Need for Software Testing :

Software bugs can cause potential monetary and human loss. There are many examples in history that clearly depicts that without the testing phase in software development lot of damage was incurred. Below are some examples :

1985 : Canada's Therac-25 radiation therapy malfunctioned due to a software bug and resulted in lethal radiation doses to patients leaving 3 injured and 3 people dead.

1994 : China Airlines Airbus A300 crashed due to a software bug killing 264 people.

1996 : A software bug caused U.S. bank accounts of 823 customers to be credited with 920 million US dollars.

1999 : A software bug caused the failure of a $1.2 billion military satellite launch.

2015 : A software bug in fighter plan F-35 resulted in making it unable to detect targets correctly.

2015 : Bloomberg terminal in London crashed due to a software bug affecting 300,000 traders on the financial market and forcing the government to postpone the 3bn pound debt sale.

Starbucks was forced to close more than 60% of its outlet in the U.S. and Canada due to a software failure in its POS system.

Nissan cars were forced to recall 1 million cars from the market due to a software failure in the car's airbag sensory detectors.

Different Types of Software Testing Techniques

Software testing techniques can be majorly classified into two categories :

1. **Black Box Testing :** Black box technique of testing in which the tester doesn't have access to the source code of the software and is conducted at the software interface without any concern with the internal logical structure of the software known as black-box testing.
2. **White-Box Testing :** White box technique of testing in which the tester is aware of the internal workings of the product, has access to its source code, and is conducted by making sure that all internal operations are performed according to the specifications is known as white box testing.
3. **Grey Box Testing :** Grey Box technique is testing in which the testers should have knowledge of implementation, however, they need not be experts.

Sr. No.	Black Box Testing	White Box Testing
1	Internal workings of an application are not required.	Knowledge of the internal workings is a must.
2	Also known as closed box/data-driven testing.	Also known as clear box/structural testing.
3	End users, testers, and developers.	Normally done by testers and developers.
4	This can only be done by a trial and error method.	Data domains and internal boundaries can be better tested.

Benefits of Software Testing :

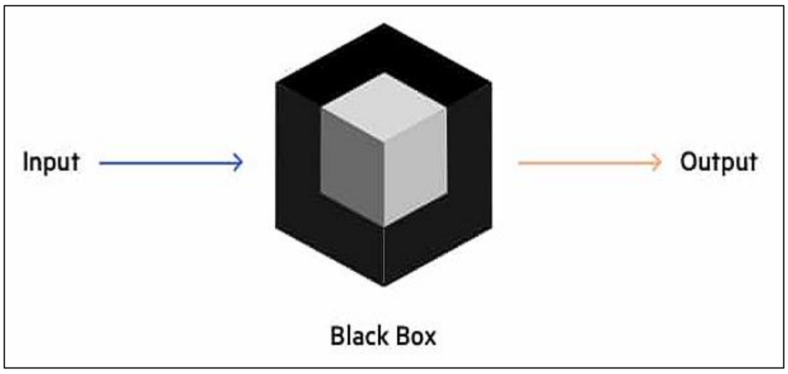
- **Product quality** : Testing ensures the delivery of a high-quality product as the errors are discovered and fixed early in the development cycle.
- **Customer satisfaction** : Software testing aims to detect the errors or vulnerabilities in the software early in the development phase so that the detected bugs can be fixed

before the delivery of the product. Usability testing is a type of software testing that checks the application for how easily usable it is for the users to use the application.

- **Cost-effective** : Testing any project on time helps to save money and time for the long term. If the bugs are caught in the early phases of software testing, it costs less to fix those errors.
- **Security** : Security testing is a type of software testing that is focused on testing the application for security vulnerabilities from internal or external sources.

5.5 Introduction to Black-Box and White Box Testing :

Black Box Testing :



Black-box testing is a type of software testing in which the tester is not concerned with the internal knowledge or implementation details of the software but rather focuses on validating the functionality based on the provided specifications or requirements. Black box testing involves testing a system with no prior knowledge of its internal workings. A tester provides an input, and observes the output generated by the system under test. This makes it possible to identify how the system responds to expected and unexpected user actions, its response time, usability issues and reliability issues.

Black box testing is a powerful testing technique because it exercises a system end-to-end. Just like end-users "don't care" how a system is coded or architected, and expect to receive an appropriate response to their requests, a tester can simulate user activity and see if the system delivers on its promises. Along the way, a black box test evaluates all relevant subsystems, including UI/UX, web server or application server, database, dependencies, and integrated systems.

Black box testing can be done in the following ways :

1. **Syntax-Driven Testing :** This type of testing is applied to systems that can be syntactically represented by some language. For example, language can be represented by context-free grammar. In this, the test cases are generated so that each grammar rule is used at least once.
2. **Equivalence partitioning :** It is often seen that many types of inputs work similarly so instead of giving all of them separately we can group them and test only one input of each group. The idea is to partition the input domain of the system into several equivalence classes such that each member of the class works similarly, i.e., if a test case in one class results in some error, other members of the class would also result in the same error.

   The technique involves two steps :

   1. **Identification of equivalence class :** Partition any input domain into a minimum of two sets : **valid values** and **invalid values**. For example, if the valid range is 0 to 100 then select one valid input like 49 and one invalid like 104.
   2. **Generating test cases –** (i) To each valid and invalid class of input assign a unique identification number. (ii) Write a test case covering all valid and invalid test cases considering that no two invalid inputs mask each other. To calculate the square root of a number, the equivalence classes will be **(a) Valid inputs :**
3. **Boundary value analysis :** Boundaries are very good places for errors to occur. Hence, if test cases are designed for boundary values of the input domain then the efficiency of testing improves and the probability of finding errors also increases. For example – If the valid range is 10 to 100 then test for 10,100 also apart from valid and invalid inputs.
4. **Cause effect graphing :** This technique establishes a relationship between logical input called causes with corresponding actions called the effect. The causes and effects are represented using Boolean graphs. The following steps are followed :
   1. Identify inputs (causes) and outputs (effect).
   2. Develop a cause-effect graph.
   3. Transform the graph into a decision table.
   4. Convert decision table rules to test cases.
5. **Requirement-based testing :** It includes validating the requirements given in the SRS of a software system.
6. **Compatibility testing :** The test case results not only depends on the product but is also on the infrastructure for delivering functionality. When the infrastructure parameters are changed it is still expected to work properly. Some parameters that generally affect the compatibility of software are :
   1. Processor (Pentium 3, Pentium 4) and several processors.

2. Architecture and characteristics of machine (32-bit or 64-bit).
3. Back-end components such as database servers.
4. Operating System (Windows, Linux, etc).

Black Box Testing Type :

The following are the several categories of black box testing :

1. Functional Testing
2. Regression Testing
3. Nonfunctional Testing (NFT)

Functional Testing : It determines the system's software functional requirements.

Regression Testing : It ensures that the newly added code is compatible with the existing code. In other words, a new software update has no impact on the functionality of the software. This is carried out after a system maintenance operation and upgrades.

Non functional Testing : Non functional testing is also known as NFT. This testing is not functional testing of software. It focuses on the software's performance, usability, and scalability.

Tools Used for Black Box Testing :

1. Appium
2. Selenium
3. Microsoft Coded UI
4. Applitools
5. HP QTP.

What can be identified by Black Box Testing :

1. Discovers missing functions, incorrect function & interface errors.
2. Discover the errors faced in accessing the database.
3. Discovers the errors that occur while initiating & terminating any functions.
4. Discovers the errors in performance or behaviour of software.

Features of black box testing :

1. **Independent testing :** Black box testing is performed by testers who are not involved in the development of the application, which helps to ensure that testing is unbiased and impartial.
2. **Testing from a user's perspective :** Black box testing is conducted from the perspective of an end user, which helps to ensure that the application meets user requirements and is easy to use.
3. **No knowledge of internal code :** Testers performing black box testing do not have access to the application's internal code, which allows them to focus on testing the application's external behaviour and functionality.

4. **Requirements-based testing :** Black box testing is typically based on the application's requirements, which helps to ensure that the application meets the required specifications.
5. **Different testing techniques :** Black box testing can be performed using various testing techniques, such as functional testing, usability testing, acceptance testing, and regression testing.
6. **Easy to automate :** Black box testing is easy to automate using various automation tools, which helps to reduce the overall testing time and effort.
7. **Scalability :** Black box testing can be scaled up or down depending on the size and complexity of the application being tested.
8. **Limited knowledge of application :** Testers performing black box testing have limited knowledge of the application being tested, which helps to ensure that testing is more representative of how the end users will interact with the application.

Advantages of Black Box Testing :

- The tester does not need to have more functional knowledge or programming skills to implement the Black Box Testing.
- It is efficient for implementing the tests in the larger system.
- Tests are executed from the user's or client's point of view.
- Test cases are easily reproducible.
- It is used in finding the ambiguity and contradictions in the functional specifications.

Disadvantages of Black Box Testing :

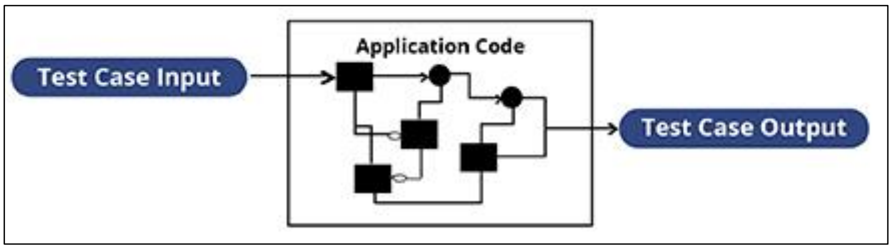
- There is a possibility of repeating the same tests while implementing the testing process.
- Without clear functional specifications, test cases are difficult to implement.
- It is difficult to execute the test cases because of complex inputs at different stages of testing.
- Sometimes, the reason for the test failure cannot be detected.
- Some programs in the application are not tested.
- It does not reveal the errors in the control structure.
- Working with a large sample space of inputs can be exhaustive and consumes a lot of time.

White Box :

White box testing techniques analyse the internal structures the used data structures, internal design, code structure, and the working of the software rather than just the functionality as in black box testing. It is also called glass box testing or clear box testing or structural testing. White Box Testing is also known as transparent testing or open box testing.

White box testing is a software testing technique that involves testing the internal structure and workings of a software application. The tester has access to the source code and uses this knowledge to design test cases that can verify the correctness of the software at the code level.

White box testing is also known as structural testing or code-based testing, and it is used to test the software's internal logic, flow, and structure. The tester creates test cases to examine the code paths and logic flows to ensure they meet the specified requirements.



White Box Testing Approach

White Box Testing is a form of penetration testing which tests internal structures of an application, as opposed to the applications functionality (also known as Black Box Testing). Programming skills are typically used to design test cases. The tester chooses inputs to exercise paths through the code and determine the expected outputs. White-box testing can be applied at three points in software testing: unit, integration and system levels.

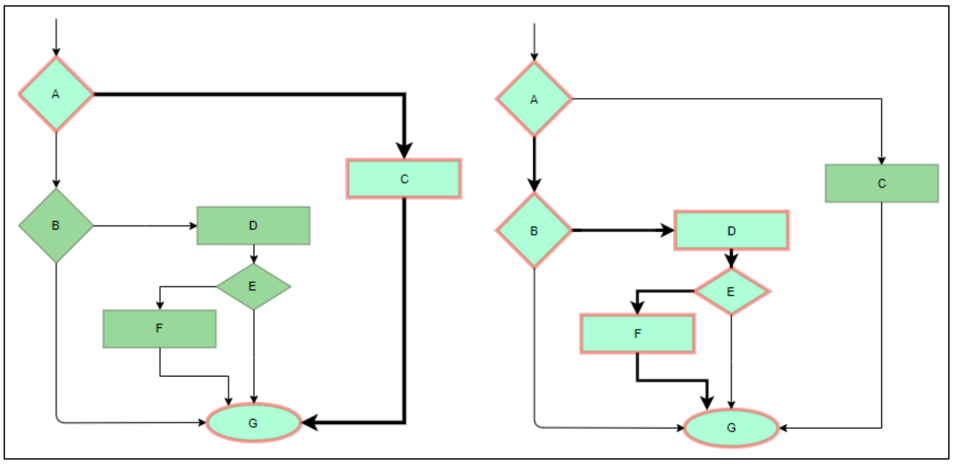
Although traditional testers tended to think of white-box testing as being done at the unit level, it is used for integration and system testing more frequently today.

Process of White Box Testing :

1. **Input :** Requirements, Functional specifications, design documents, source code.
2. **Processing :** Performing risk analysis to guide through the entire process.
3. **Proper test planning :** Designing test cases to cover the entire code. Execute rinse-repeat until error-free software is reached. Also, the results are communicated.
4. **Output :** Preparing final report of the entire testing process.

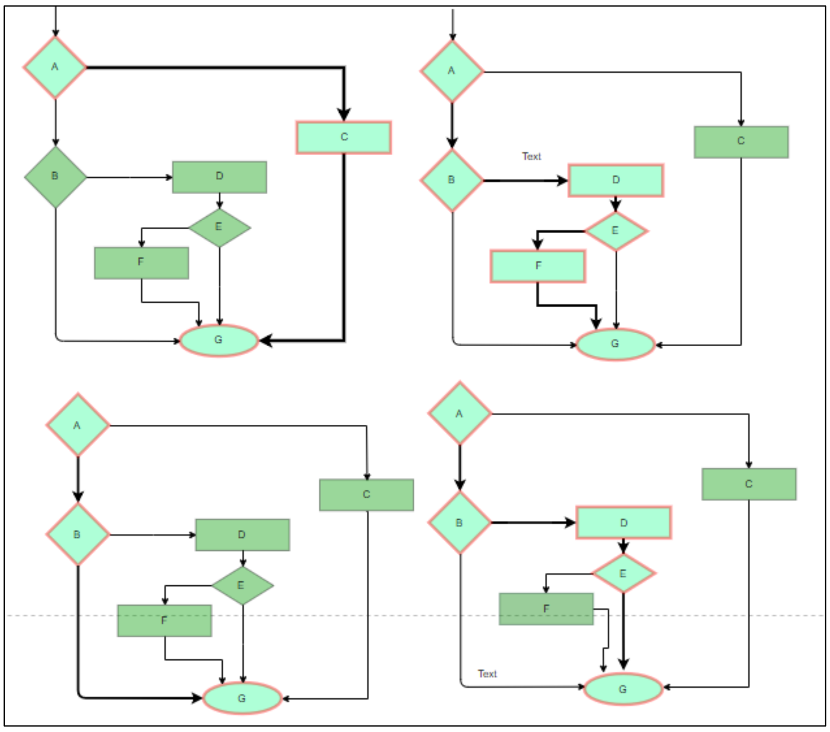
Testing Techniques :

1. **Statement Coverage :** In this technique, the aim is to traverse all statements at least once. Hence, each line of code is tested. In the case of a flowchart, every node must be traversed at least once. Since all lines of code are covered, it helps in pointing out faulty code.



Statement Coverage Example

2. **Branch Coverage** : In this technique, test cases are designed so that each branch from all decision points is traversed at least once. In a flowchart, all edges must be traversed at least once.



4 test cases are required such that all branches of all decisions are covered, i.e, all edges of the flowchart are covered

3. **Condition Coverage** : In this technique, all individual conditions must be covered as shown in the following example:
   - READ X, Y
   - IF(X == 0 || Y == 0)
   - PRINT ‘0’
   - #TC1 – X = 0, Y = 55
   - #TC2 – X = 5, Y = 0
4. **Multiple Condition Coverage** : In this technique, all the possible combinations of the possible outcomes of conditions are tested at least once. Let’s consider the following example :
   - READ X, Y
   - IF(X == 0 || Y == 0)
   - PRINT ‘0’
   - #TC1: X = 0, Y = 0
   - #TC2: X = 0, Y = 5
   - #TC3: X = 55, Y = 0
   - #TC4: X = 55, Y = 5
5. **Basis Path Testing** : In this technique, control flow graphs are made from code or flowchart and then Cyclomatic complexity is calculated which defines the number of independent paths so that the minimal number of test cases can be designed for each independent path.

   Steps :
   - Make the corresponding control flow graph.
   - Calculate the cyclomatic complexity.
   - Find the independent paths.
   - Design test cases corresponding to each independent path.
   - V(G) = P + 1, where P is the number of predicate nodes in the flow graph.
   - V(G) = E – N + 2, where E is the number of edges and N is the total number of nodes.
   - V(G) = Number of non-overlapping regions in the graph.
   - #P1: 1 – 2 – 4 – 7 – 8.
   - #P2: 1 – 2 – 3 – 5 – 7 – 8.
   - #P3: 1 – 2 – 3 – 6 – 7 – 8.
   - #P4: 1 – 2 – 4 – 7 – 1 – . . . – 7 – 8.
6. **Loop Testing :** Loops are widely used and these are fundamental to many algorithms hence, their testing is very important. Errors often occur at the beginnings and ends of loops.

- **Simple loops :** For simple loops of size n, test cases are designed that :
    1. Skip the loop entirely
    2. Only one pass through the loop
    3. 2 passes
    4. m passes, where m < n
    5. n-1 ans n+1 passes
- **Nested loops :** For nested loops, all the loops are set to their minimum count, and we start from the innermost loop. Simple loop tests are conducted for the innermost loop and this is worked outwards till all the loops have been tested.
- **Concatenated loops :** Independent loops, one after another. Simple loop tests are applied for each. If they're not independent, treat them like nesting.

Advantages of White box Testing :

1. **Thorough Testing** : White box testing is thorough as the entire code and structures are tested.
2. **Code Optimization :** It results in the optimization of code removing errors and helps in removing extra lines of code.
3. **Early Detection of Defects :** It can start at an earlier stage as it doesn't require any interface as in the case of black box testing.
4. **Integration with SDLC :** White box testing can be easily started in Software Development Life Cycle.
5. **Detection of Complex Defects :** Testers can identify defects that cannot be detected through other testing techniques.
6. **Comprehensive Test Cases :** Testers can create more comprehensive and effective test cases that cover all code paths.
7. Testers can ensure that the code meets coding standards and is optimized for performance.

Disadvantages of White box Testing :

1. **Programming Knowledge and Source Code Access :** Testers need to have programming knowledge and access to the source code to perform tests.
2. **Overemphasis on Internal Workings :** Testers may focus too much on the internal workings of the software and may miss external issues.
3. **Bias in Testing :** Testers may have a biased view of the software since they are familiar with its internal workings.
4. **Test Case Overhead :** Redesigning code and rewriting code needs test cases to be written again.
5. **Dependency on Tester Expertise :** Testers are required to have in-depth knowledge of the code and programming language as opposed to black-box testing.

6. **Inability to Detect Missing Functionalities :** Missing functionalities cannot be detected as the code that exists is tested.
7. **Increased Production Errors :** High chances of errors in production.

: QUESTIONS :

1. Discuss user – interface design in brief.
2. Discuss Graphical interfaces in brief.
3. Write short note on Elements of Good Interface Design.
4. What do you mean by software testing? What are advantages of software testing?
5. What is Black box testing? State any four black box testing.
6. State any three advantages and disadvantages of black box software testing.
7. What is Black box testing? State any two black box testing, shortly.
8. State any three advantages and disadvantages of white box software testing.
9. Differentiate black box and white box testing.

***

6. Chapter

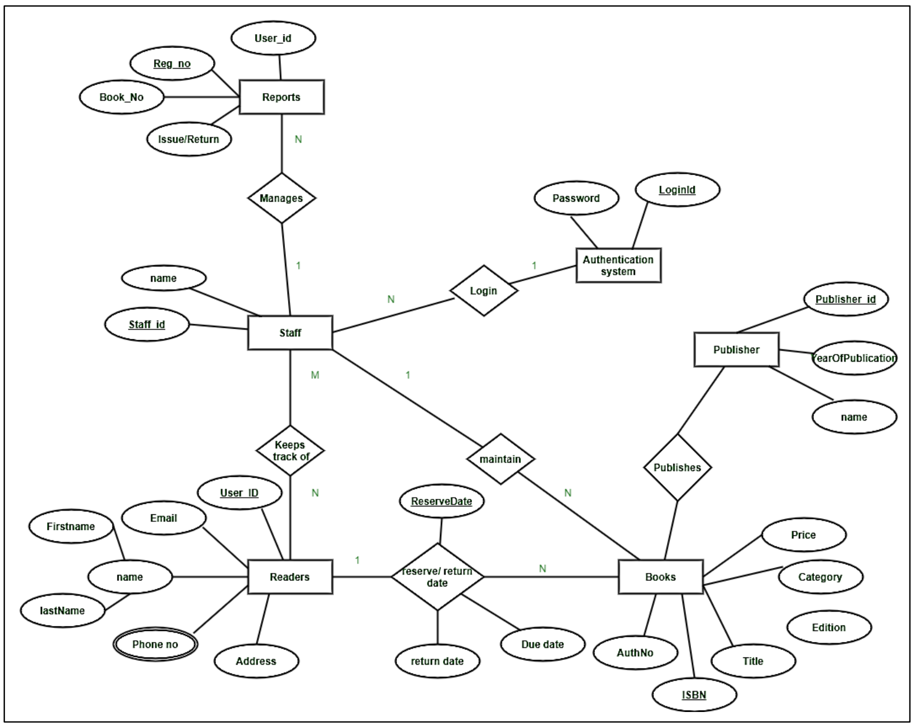
Designing business application system using DFD, ERD

C o n t e n t s :

(10 L, 15 M)

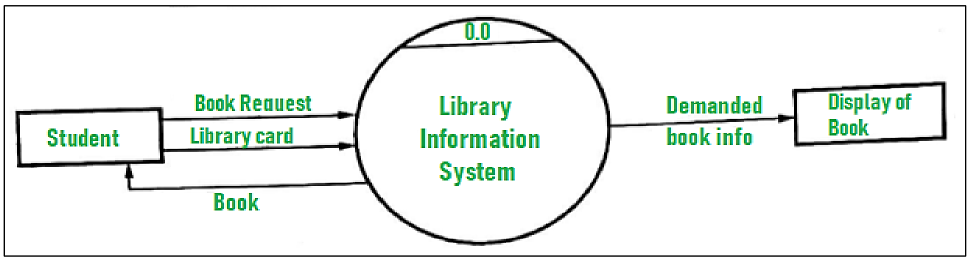
6.1 Library Management System :

ER Diagram :

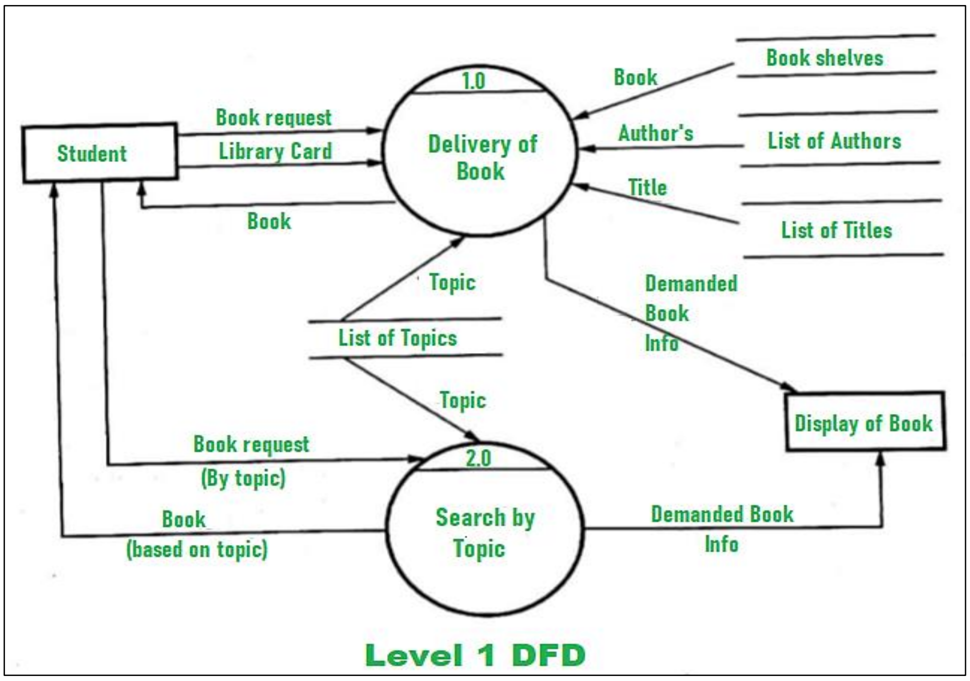


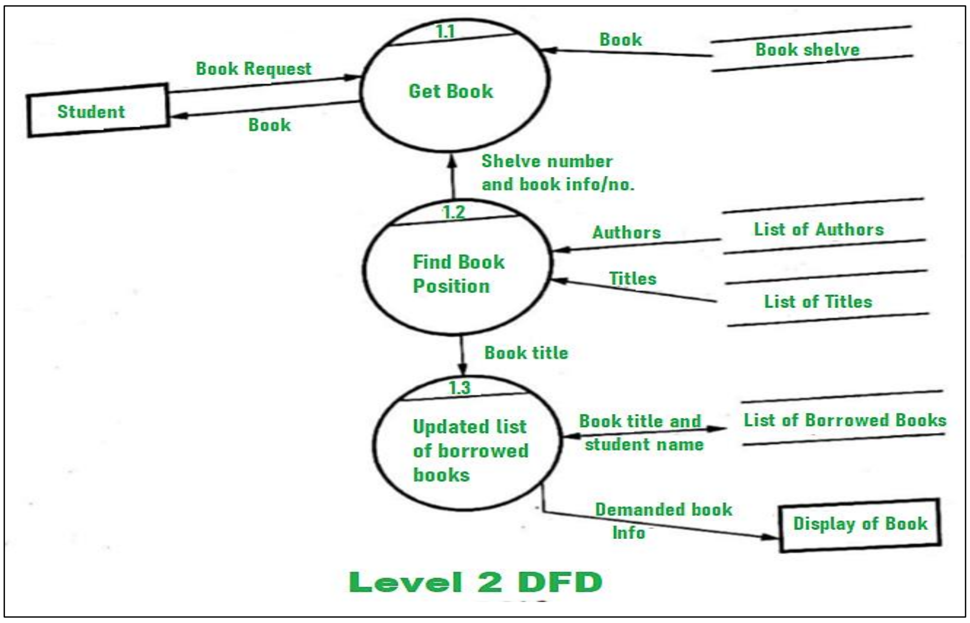
Data Flow Diagram (DFD) :

Example of DFD level 0, Level 1 and Level 2 :



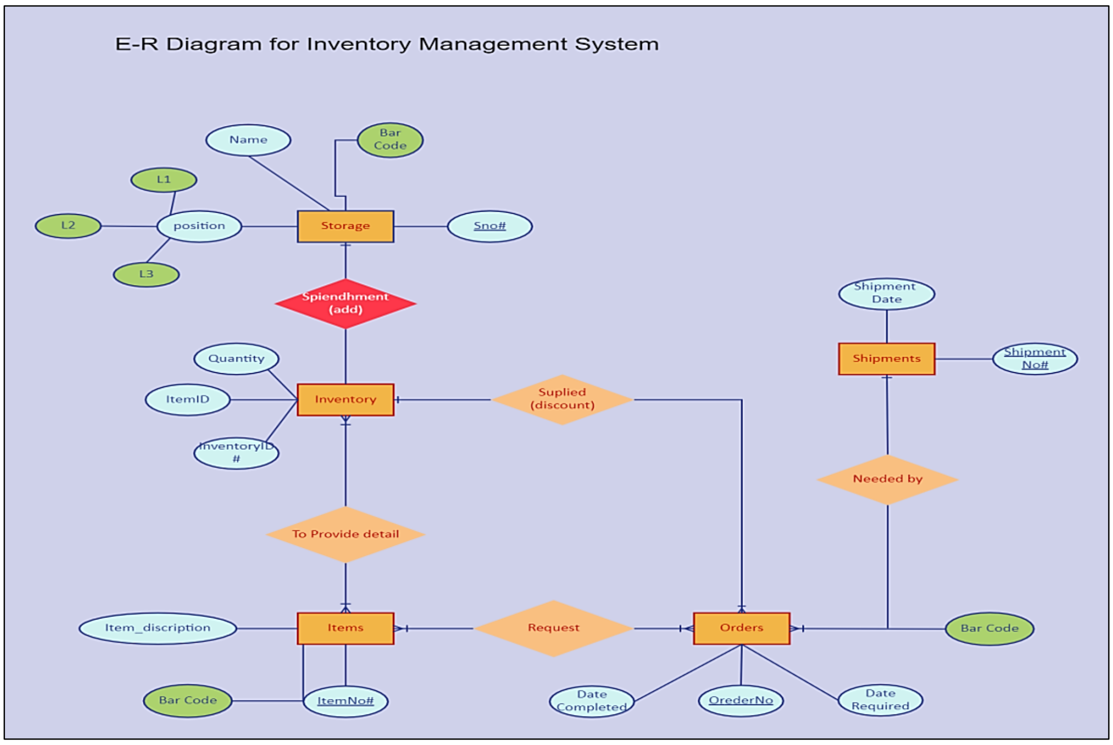
Context Level DFD or Level 0 DFD



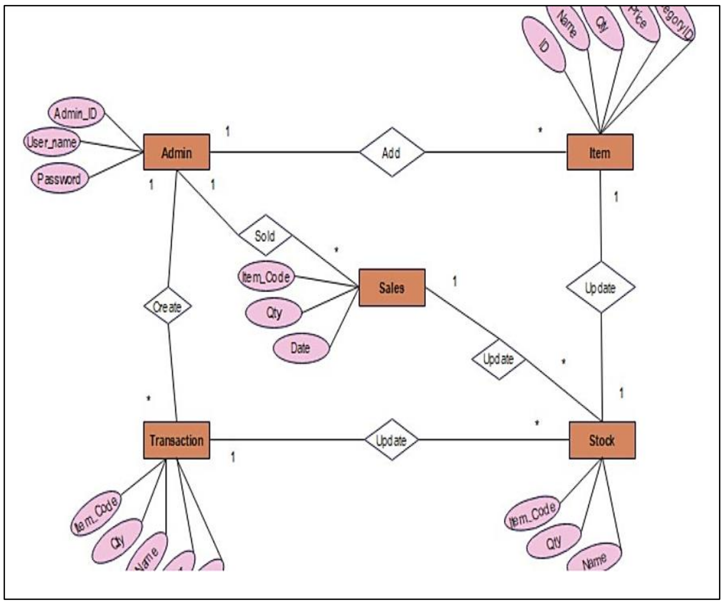


6.2 Inventory Management System :

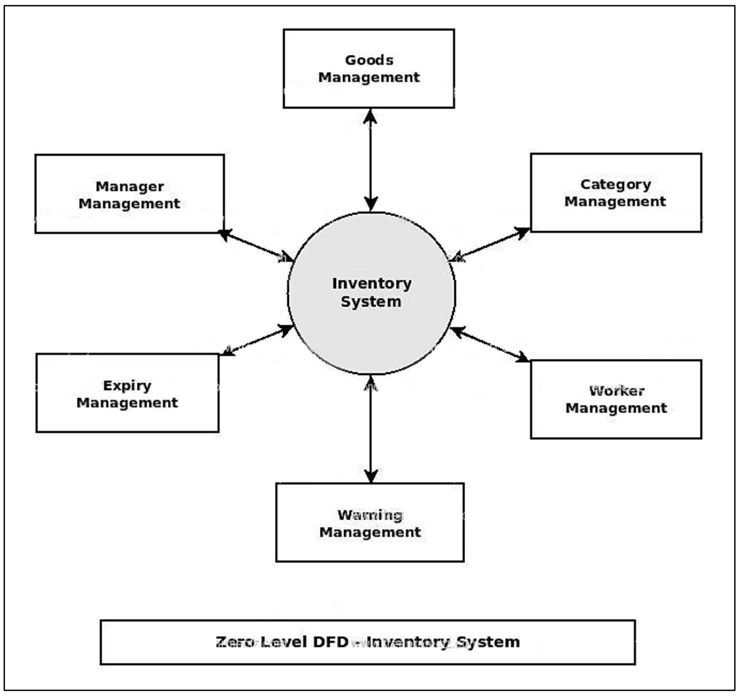
ER Diagram :

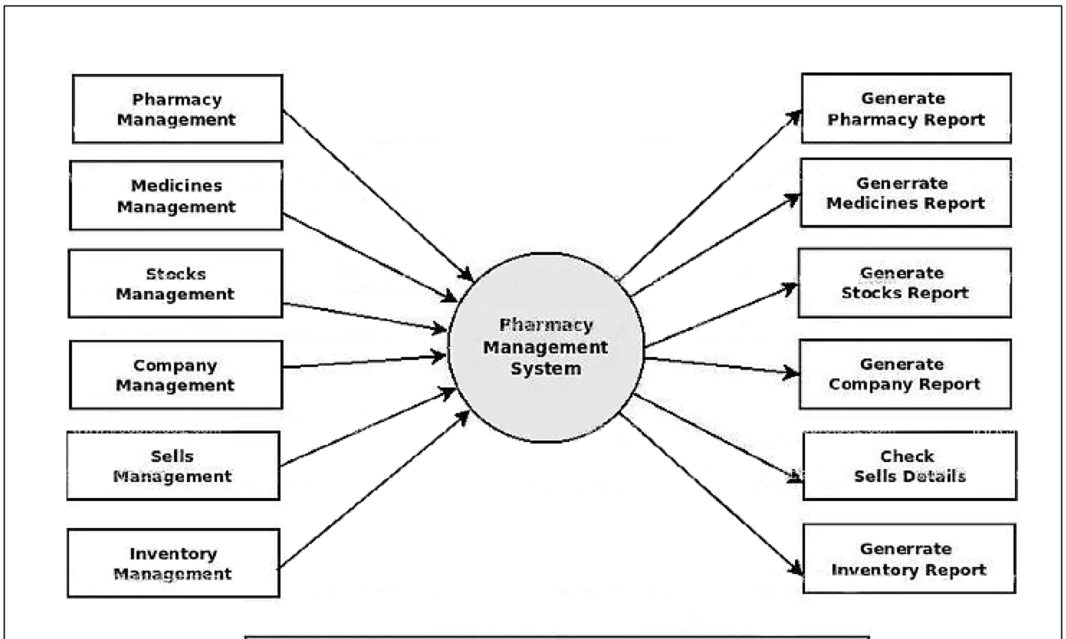


ER Diagram for Inventory Management :

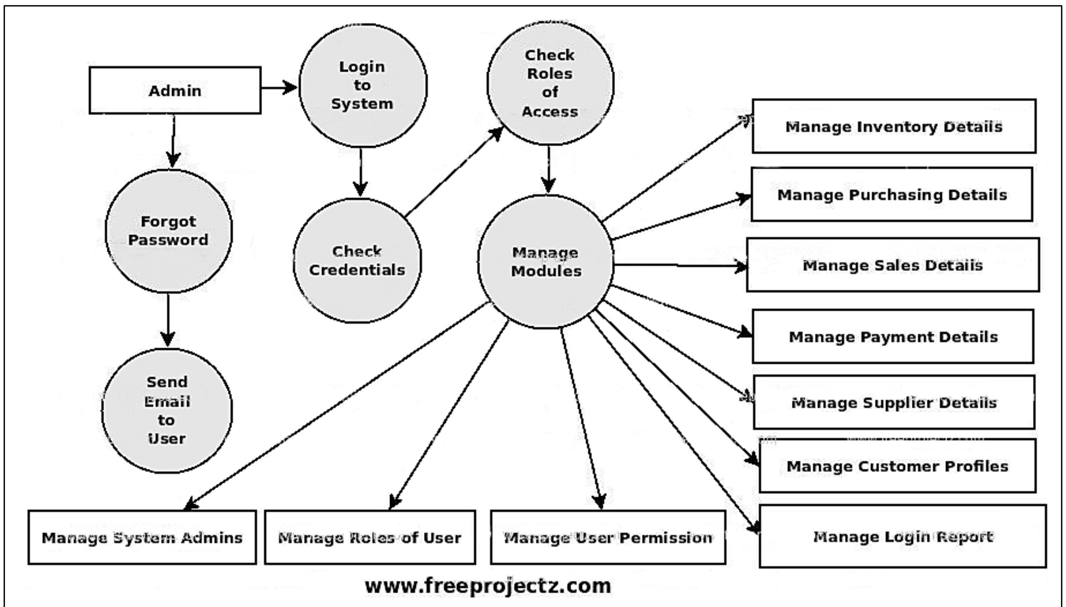


Data Flow Diagram (DFD) :





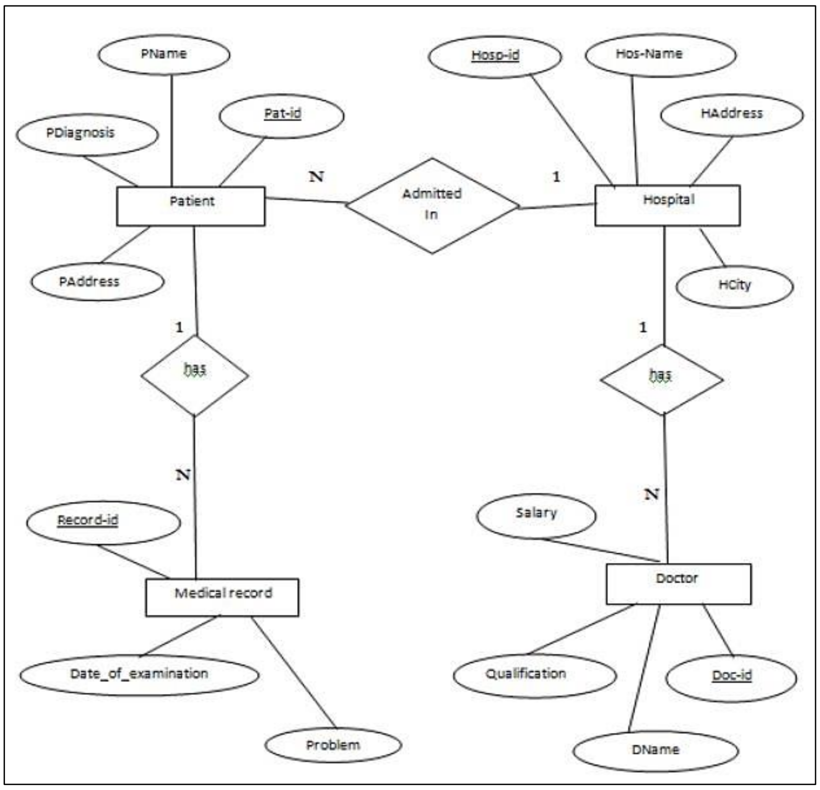
First Level DFD - Pharmacy Management System



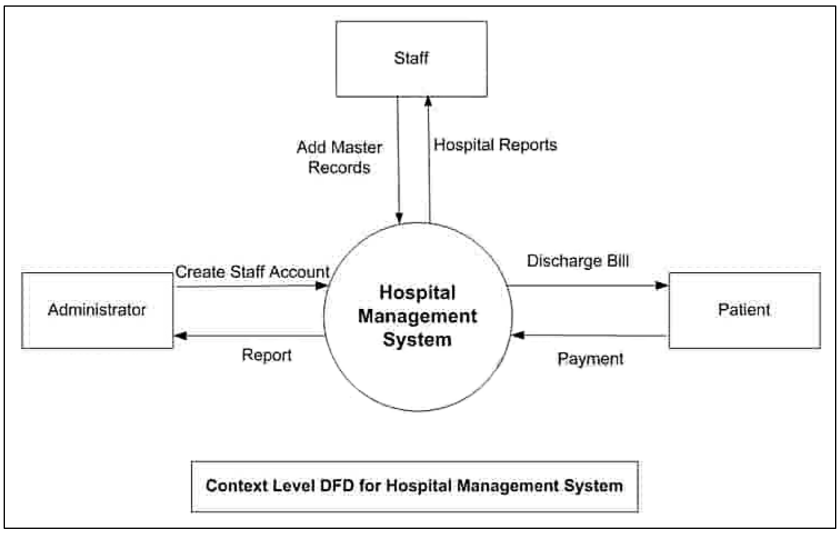
Second Level DFD - Sales And inventory Management System

6.3 Hospital Management System :

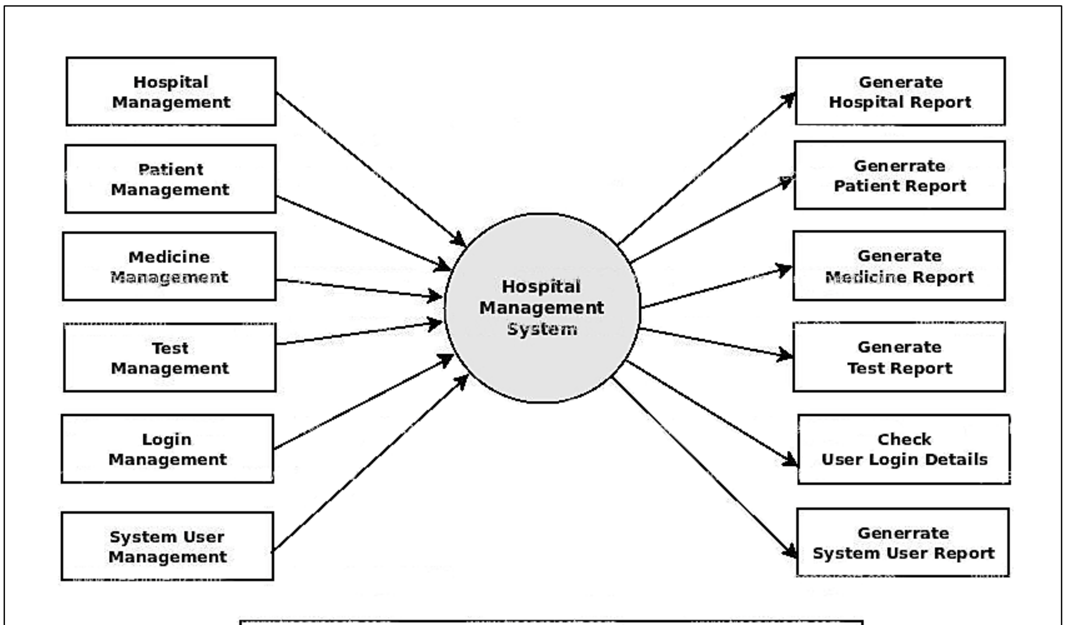
ER Diagram :



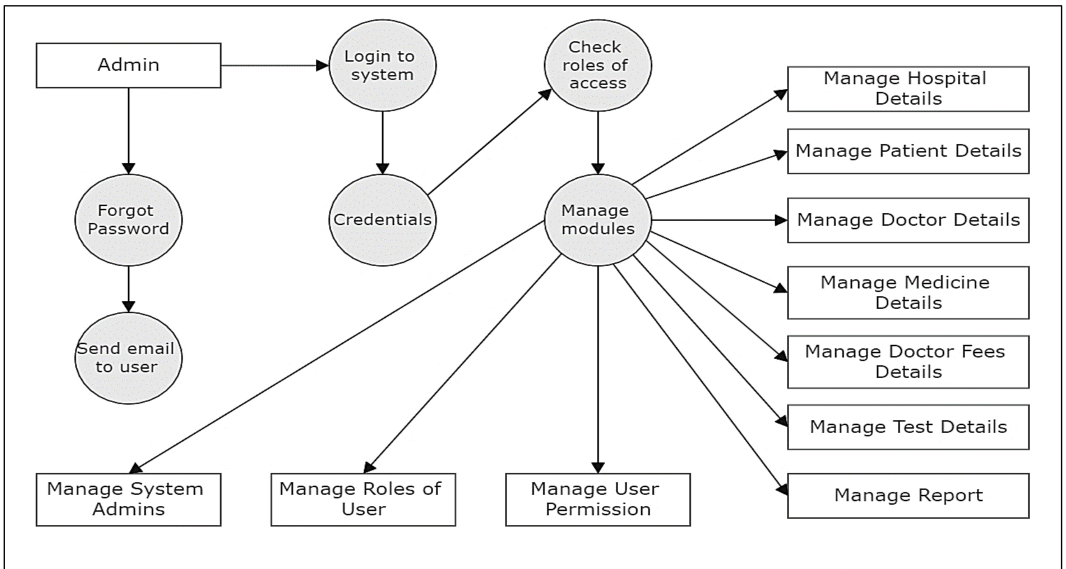
Data Flow Diagram :



Context Level DFD for Hospital Management System



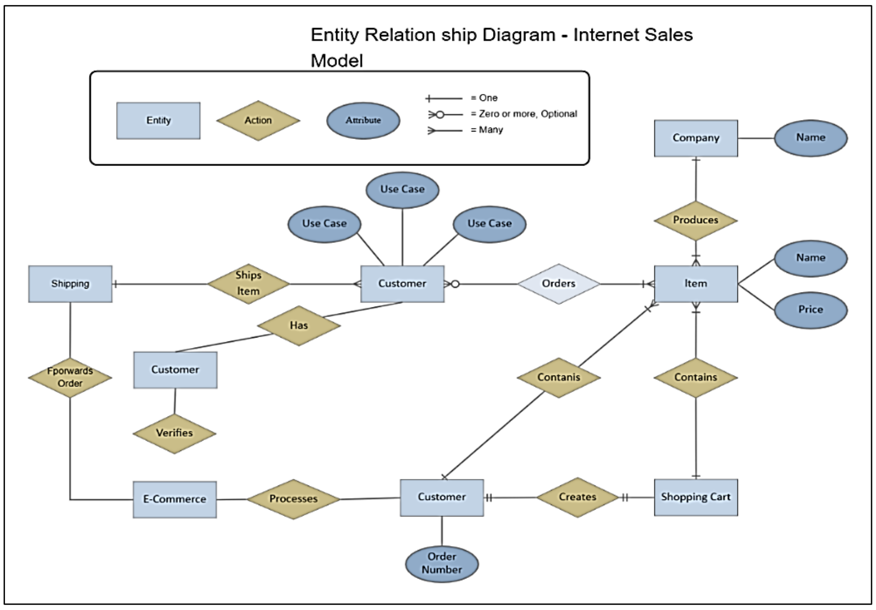
First Level DFD - Hospital Management System



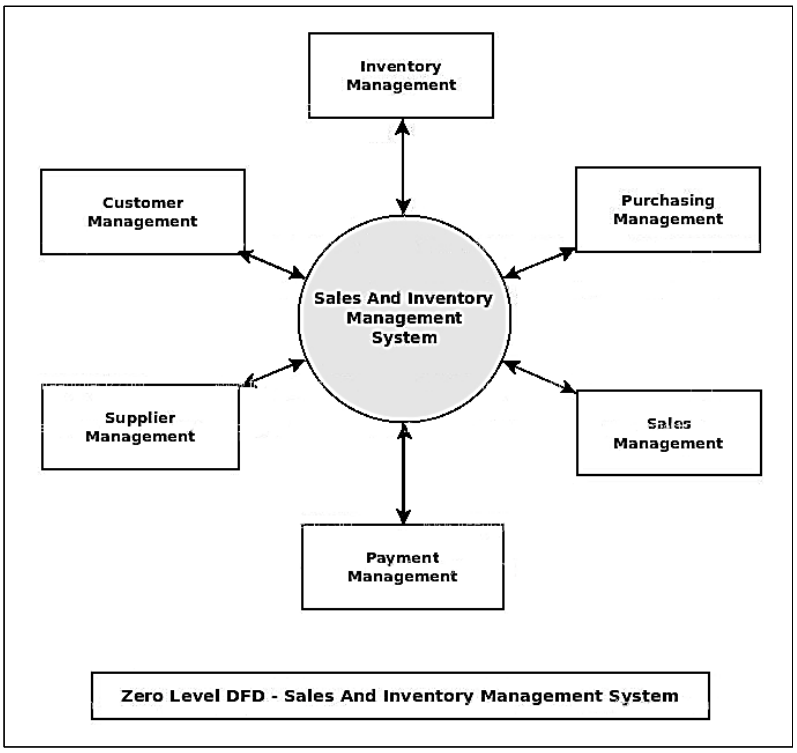
Second Level DFD – Hospital Management System

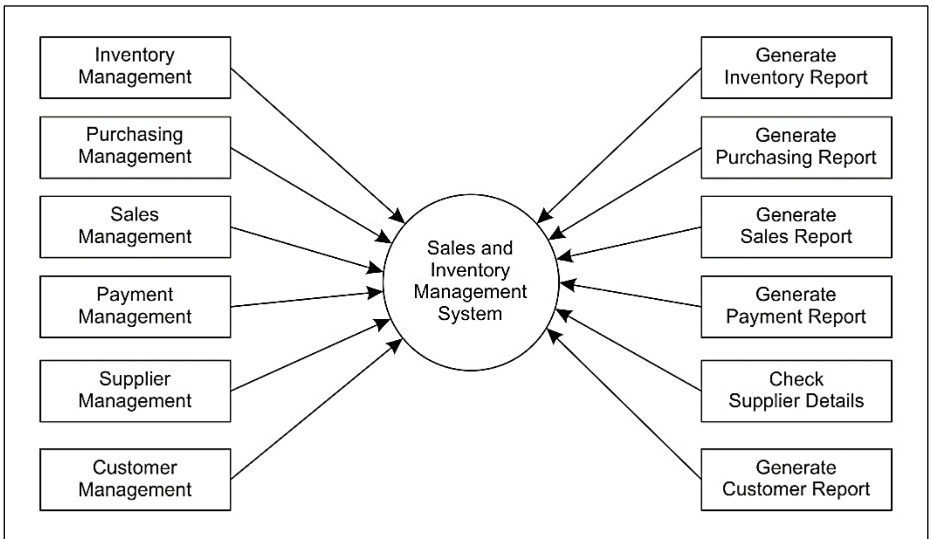
6.4 Sales / Purchases System :

ER Diagram :

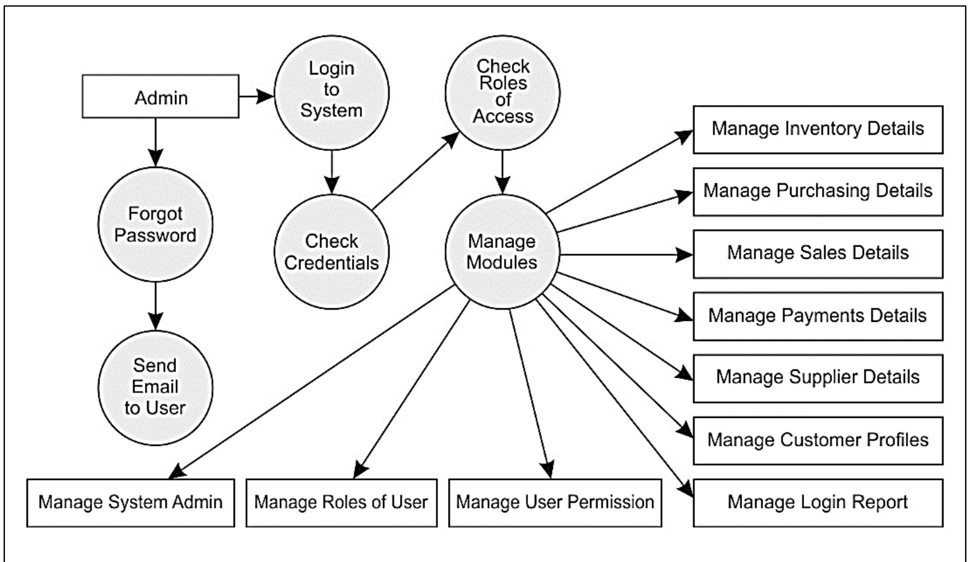


Data Flow Diagram :





First Level DFD - Sales and Inventory Management System



Second Level DFD - Sales and Inventory Management System

***

As Per New Syllabus of Kavayitri Bahinabai Chaudhari
North Maharashtra University, Jalgaon

Bachelor of Computer Application (BCA) (w.e.f. 2023-24)

SEMESTER I	
BCA 101	Fundamentals of Accounting
BCA 102	Fundamentals of Computer
BCA 103	Programming in C – I
BCA 104	Web Design – I

SEMESTER II	
BCA 201	Professional Communication Skill
BCA 202	Database Management System
BCA 203	Programming in C – II
BCA 204	Web Design - II

SEMESTER III	
BCA 301	Fundamental Mathematics and Statistics
BCA 302	Operating System
BCA 303	Programming in C++
BCA 304A	Web Development Technology – I
BCA 304C	Python Programming

SEMESTER IV	
BCA 401	Software Engineering
BCA 402	Data Structure
BCA 403	Java Programming
BCA 404A	Web Development Technology - II
BCA 404C	Artificial Intelligence

SEMESTER V	
BCA 501	Employability Skill
BCA 502	E-Commerce and M-Commerce
BCA 503	Cloud Computing Application
BCA 504A	Web Development Technology – III
BCA 504C	Machine Learning

SEMESTER VI	
BCA 601	Entrepreneurship Development
BCA 602	Cyber Security
BCA 603	Android Application Development
BCA 604A	Web Development Technology – IV
BCA 604C	Data Mining

Prashant

BCA ₹ 60

ISBN 978-81-19120-92-5

www.prashantpublications.com
prashantpublication.jal@gmail.com