SPPU New Syllabus

A Book Of

SOFTWARE ENGINEERING

For BBA (Computer Applications): Semester - III

[Course Code CA-303 : Credit - 4]

&

B.C.A.(Science) : Semester – IV

[Course Code BCA-243 : Credit - 4]

CBCS Pattern

As Per New Syllabus, Effective from June 2020

Mrs. Manisha Surywanshi
M.C.S., M.Phil (Comp. Sci.). M.B.A. (HR)
Placement Co-ordinator,
Assistant Professor,
Modern College of Arts, Science
and Commerce (Autonomous)
Pune 5.

Dr. A. B. Nimbalkar
M.C.S., M.Phil. (C.S.), Ph.D.(Comp. Sci.), D.C.L
Assistant Professor,
Annasaheb Magar Mahavidyalaya,
Hadapsar,
Pune 28.

Price ₹ 270.00

NIRALI PRAKASHAN
ADVANCEMENT OF KNOWLEDGE

N4949

Syllabus ...

B.B.A. (C.A.) SEMESTER – III
COURSE CODE: CA-303

1.1 Definition
1.2 Basic Components
1.3 Elements of the System
1.4 Types of System
1.5 System Characteristics

2.1 Definition of Software
2.2 Characteristics of Software
2.3 Definition of Software Engineering
2.4 Need for Software Engineering
2.5 Mc Call's Quality factors
2.6 The Software Process
2.7 Software Product and Process
2.8 V& V Model

3.1 Introduction
3.2 Activities of SDLC
3.3 A Generic Process Model
3.4 SDLC
3.5 Waterfall Model
3.6 Incremental Process Models
3.7 Prototyping Model
3.8 Spiral Model

4.1 Introduction

4.2 Requirement Elicitation
4.3 Requirement Elaboration
4.4 Requirement Gathering
4.5 Feasibility study
4.6 Fact Finding Techniques
4.7 SRS Format

5.1 Decision Tree and Decision Table
5.2 Data Flow Diagrams (DFD) (Up to 2nd level)
5.3 Data Dictionary
5.4 Elements of DD
5.5 Advantages and Disadvantages of DD
5.6 Input and Output Design
5.7 Structured Design Concepts
5.8 Structure Chart
5.9 Coupling and Cohesion
5.10 Compulsory Case Studies on above topics

6.1 Definition
6.2 Software testing Process
6.3 Unit Testing
6.4 Integration Testing
6.5 System Testing

7.1 Maintenance definition and types
7.2 Software reengineering
7.3 Reverse Engineering
7.4 Restructuring and forward Engineering.

❖❖❖

BCA (SCIENCE) SEMESTER–IV
COURSE CODE: BCA-243

Unit I Introduction to System concepts

1.1 Definition
1.2 Basic Components
1.3 Elements of the system
1.4 System Components
1.5 Types of System

Unit II Introduction to Software Engineering

2.1 Definition of Software
2.2 Characteristics of Software
2.3 Software Application Domains
2.4 Definition of Software Engineering
2.5 Need for software Engineering
2.6 Mc Call's Quality factors
2.7 The Software Process
2.8 Software Engineering Practice

Unit III Software Development Life Cycle (SDLC) and Models

3.1 Introduction
3.2 Activities of SDLC
3.3 A Generic Process Model
3.4 Prescriptive Process models
    3.4.1 Waterfall Model
    3.4.2 Incremental Process Models
    3.4.3 Evolutionary process Models (Prototyping and Spiral Model)
3.5 Concurrent Models, Types

Unit IV Requirement Engineering

4.1 Introduction
4.2 Requirement Engineering Tasks
4.3 Establishing Groundwork for understanding of Software Requirement
4.4 Requirement Gathering
4.5 Feasibility study
4.6 Fact Finding Techniques

1.2 DEFINITION OF SYSTEM

- A system is an orderly grouping of interdependent components linked together according to a plan to achieve a specific goal/objective.
- A system is defined as, "a collection of related components that interact to perform a task in order to accomplish a goal".

OR

- A system can be defined as, "an integrated collection of components which satisfy functions (tasks) necessary to achieve the system's objectives or goals".
- There are following basic implications of any system as given below:
  1. A system must be **designed to achieve pre-determined objectives.**
  2. A system must have some **structure and behavior** which is designed to achieve a predefined objective.
  3. In any system **interconnectivity and interdependence** must exist among the system components.
  4. A system must be **user friendly** and it should **maintain security** from unauthorized users as well as making selected data.
  5. The objectives of the system have a **higher priority** than the objectives of its sub-systems.

1.3 CHARACTERISTICS OF A SYSTEM (BCA:W-18; BBA(CA):W-18)

Following are important characteristics of a system:

1. **Organization**
- Organization implies structure and order.
- It is the arrangement of components that helps to achieve pre-determined objectives/goals.
- For example, a computer system is designed around an input device, a central processing unit and an output device with one or more storage units, when linked together they work as a whole system for producing information.

2. **Interaction**
- It is defined by the manner in which the components are operate with each other.
- For example, in an organization, purchasing department must interact with production department and payroll with personnel department.

3. **Interdependence**
- Interdependence means how the components of a system depend on one another.
- For proper functioning, the components are co-ordinated and linked together according to specified plan.

- The output of one sub-system is the required by other sub-system as input.
- For example, in computer system, the three units: Input unit, System unit and Ou unit are interdependent for proper functioning. No subsystems can functio isolation because it is dependent on data (input) it receives from other subsystem perform its required tasks.

4. **Integration**
- Integration is concerned with how a system is tied together in order to achi common goal, thus forming integration.
- It means that the parts of the system work together within the system even if e part performs a unique function.

5. **Central Objective**
- The objective of system must be central.
- Central objective means the common goal and it may be real or stated. It is uncommon for an organization to state an objective and operate to achieve another
- The users must know the central objective of a computer application in the analy for a successful design and conversion.

6. **Behavior:**
- Behavior is the way the system reacts to its surrounding environment.
- Behavior is determined by the procedures designed to make sure that compone behave in ways that will allow system to achieve common goal.

7. **Structure**
- A relationship among the components which define the boundary between system and environment is called as the structure of the system.

1.4 BASIC COMPONENTS OF A SYSTEM

- The functioning units of a system mean the basic elements of the system which interrelated. These are the basic components of the system.
- Every system is made up of a set of interrelated elements or basic components. Th components are the various parts of a system.
- For examples:

Sr. No.	System	Components
1.	Education	Students, Teachers, Library, Laboratory, Buildings, etc.
2.	Computer	Mouse, Display Unit, ALU, Hard Disk, Keyboard, etc.

- All systems having com
- Those basic component
- Fig. 1.1 shows followin

Fig. 1.1: C

1. Input
- A main objective of
- Inputs are elemen output.
- Inputs a provide
- Input is d
- It may be r output.

output.
- The fcr
- Someti transforma

3. Output
- A major objecti
- Though output may be differe
- Output is defi
- In other word new knowledg
- Fig. 1.2 shows

- All systems having common main three components in which all they are described.
- Those basic components are input, processor and output as shown in Fig. 1.1.
- Fig. 1.1 shows following basic components of the system:

Fig. 1.1: General Model of a System with its Basic Components

1. **Input**

- A main objective of a system is to produce the output as per the user's requirement.
- Inputs are elements that make the system to work in order to produce required output.
- Inputs are the elements that enter the system for processing. The inputs to the system provide all the needed resources to accomplish the goals of the system.
- Input is defined as, "energizing or start up component on which system operates".
- It may be raw material, data, physical source, knowledge etc., to decide the nature of output.

2. **Processor/Process**

- The processes are the action (operation or task) which brings about system goals using the inputs (resources).
- Processor is the element that involves in the actual the transformation of input to output.
- The processor should be designed of such type that it can accept the input in the given form and can give output in desired format or specification.
- Sometime, input is also modified to enable the processor so that it can be handle transformation easily. This modification may be total or partial.

3. **Output**

- A major objective of a system is to produce the output as per the user's requirement.
- Though output largely depends upon the input, its nature, amount, utility, regularity may be different from those of input.
- Output is defined as, "the result of an input operation after processing".
- In other words, the outputs of the system include the goals (products or services or new knowledge etc.) which the system is designed.
- Fig. 1.2 shows an example of the Basic System Model (Payroll System).

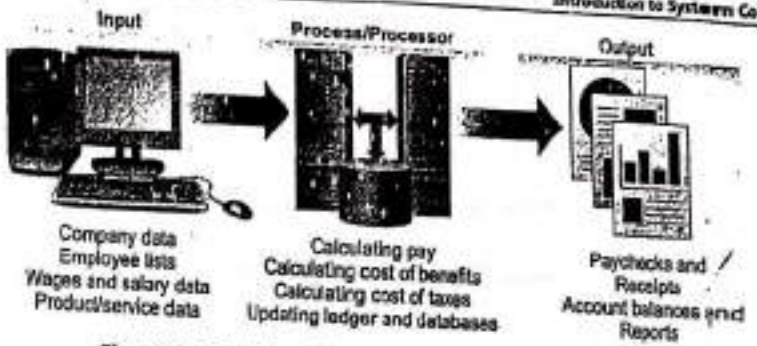


Fig. 1.2: The Basic Payroll System with its Basic Components

- These basic components have certain properties or characteristics. These characteristics affect the operation of the system in speed, accuracy, reliability and capacity.

1.5 ELEMENTS OF SYSTEM (BCA: S-18, W-18; BBA(CA): W-18, S-19)

- All the characteristics of the system are determined by the system elements, their properties and relationships.
- The basic system elements are Input, Processor and Output (Explained in Section 1.3).
- Universal model of a system is made up of system elements like input, processor and output using basic concept like control and feedback to keep the system balance.
- The universal system model is as shown in Fig. 1.3.

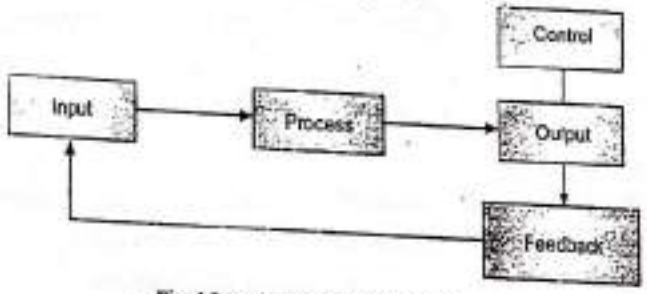


Fig. 1.3: Universal Model of a System

- Each system processes a model and the model is made up of following system elements:

1. **Inputs and Outputs:**

- Inputs are the information that enters into the system for processing while Output is the outcome of processing.

2. **Processor(s):**

- It is the operational component of a system and involves the actual transformation of input into output.

3. **Control:**

- The control elements guide the system. It controls the working of the system at all stages.
- It controls all the input, processor and output activities so that decision can be taken property.

4. **Feedback:**

- Feedback is a method that helps to compare output produced with output expected and make necessary changes in the process or input in order to reduce the difference between output produced and output expected.
- It is helpful for decision making in any system. On the basis of feedback the new business strategies were decided in any organization.

Some other System elements:

- Some other system elements are essential for any system building are explained below and shown in Fig. 1.4.

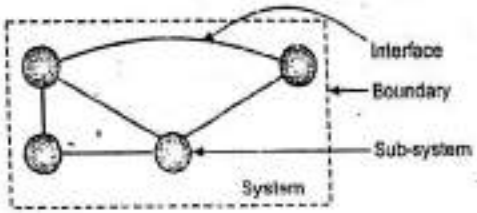


Fig. 1.4: Other Essential Elements of a System

1. **Environment:**

- All the things which are outside the system are called as environment of the system.
- The environment affects working or progress of the system.
- It is a super-system in which any organisation operates.

2. **Boundaries:**

- The boundary indicates the extent or limit of the system. It divides things into the system and its environment.
- The elements of the system design its boundary.
- It is very much essential to limit the system by its boundaries so that system's working can be controlled.

3. **Interface:**

- Interface means interaction of different system parts with each other as interaction of the system with the system outside its boundaries.

4. **Sub-system:**

- Sub-system is a unit that is a part of a larger system. That means the larger system divided into sub-parts. These sub-parts are again divided into smaller sub-systems until the smallest sub-systems are of manageable size.
- For example, computer is system and sub-system contains CPU, Input Unit, Output Unit, and so on.

5. **Black Box System:**

- A sub-system at the lowest level whose processes are not defined is called Black Box system.
- Only inputs and outputs are known but not actual transformation.

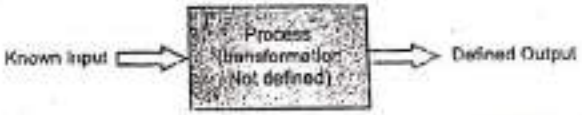


Fig. 1.5: A Black Box System

1.6 TYPES OF SYSTEMS (BCA: S-18, 19; BBA(CA): W-18, S-

- The system is classified in different types on the basis of its analysis. Some types systems are explained below:

1. **Conceptual (abstract) and Physical Systems**

- The conceptual system is also known as abstract or analytical system. It is n physical system.
- **Abstract Systems** are represented conceptually (i.e. not existing) non-physi systems are called abstract system.
- These systems are prepared for studying the physical system.
- For example, the computer itself is a physical system and its block diagram is ca as abstract system.
- These systems may be formulas, representation or model of real system.
- The **Physical Systems** are tangible (countable entities) and could be static or dyna in nature. Static means which do not change as for working or life of the syste concerned. On the other hand, the term dynamic means, system can be change du processing of the system.

- For example, in computer system the hardware parts are static but the data which changes due to processing is dynamic.

2. **Deterministic and Probabilistic Systems**

- A deterministic system works in predictable manner.
- A Deterministic system is one in which the occurrence of all events is perfectly predictable. If we get the description of the system state at a particular time, the next state can be easily predicted.
- An example of a deterministic system is the common entrance examination for entry into IIM. All the entities in the system and their interrelationships are well known and given an input the output can be determined with certainty.
- For example, in computer system, outputs are deterministic.
- A probabilistic System is one in which the occurrence of events cannot be perfectly predicated.
- Though the behavior of such a system can be described in terms of probability, a certain degree of error is always attached to the prediction of the behaviour of the system.
- For example, Weather forecasting system is probabilistic.

3. **Open and Closed Systems**

- Another classification of system is based on their degree of independence.

(i) **Open System:**

- An open system is a state of a system, in which a system continuously interacts with its environment.
- It visualizes organizations taking inputs then operations are performed on the input to produce desirable results (output) which are distributed back to the environment.

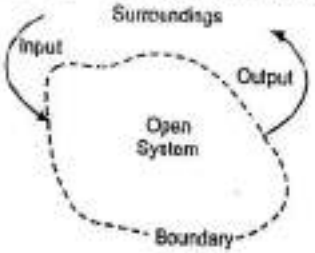


Fig. 1.6: An Open System

- An open system is a one which does not provide for its own control or modification. It does not supervise itself so it needs to be supervised by people.

For example: If the high speed printer used with computer systems did not have a switch to sense whether paper is in the printer, then a person would have to notice when the paper runs out and signal the system (push a switch) to stop printing.

- Open systems have input and output flows, representing exchanges of matter, energy or information with their surroundings.
- For example, in education syutem, student's information is shared with companies for placement.
- Open systems have many interfaces with environment. These system permits interaction across its boundary

Characteristics:

- Five important characteristics of open system are:
  - (a) **Input from outside:** Open systems are self adjusting and self regulating. When functioning properly, an open system reaches a steady state. Open system can accept input from outside also.
  - (b) **Entropy:** Entropy means loss of energy. All dynamic system tends to run down over time resulting in entropy. Open system result entropy by seeking new inputs to return to a steady state.
  - (c) **Process, Output and Cycle:** Open system produce useful output and operate in cycles following a continuous flow path.
  - (d) **Differentiation:** Open system always increase specialization of functions and greater differentiation of their components. This characteristic offers a completing reason for increasing value of concept of system in system analyst's thinking.
  - (e) Equifinality: This term implies that the goals are achieved through differing courses of action and a variety of paths. For example, teaching is the goal can be achieved by giving black-board teaching or PowerPoint presentation.

(ii) Closed System:

- A system which does not interact with the outside environment is known as the closed system i.e. it has no input or output.

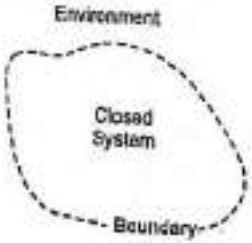
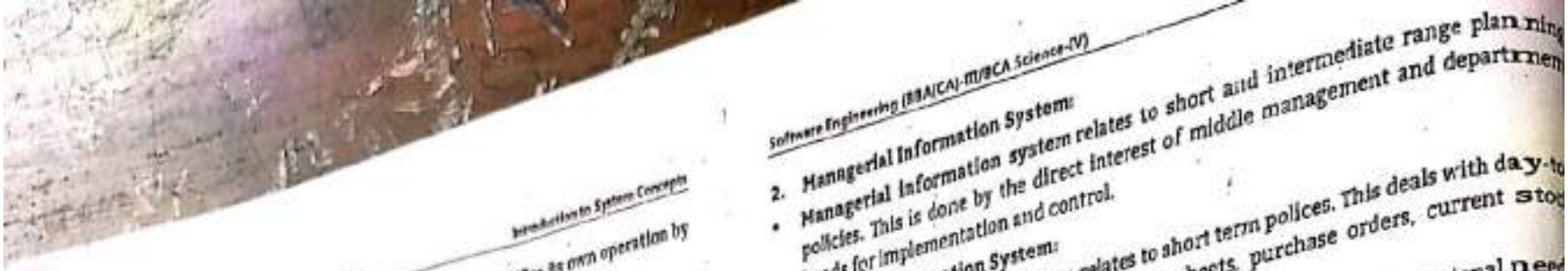


Fig. 1.7: Closed System



- A closed system is one which automatically controls or modifies its own operation by responding to data generated by the system itself.
- A closed system is a system in the state of being isolated from the environment.
- For example, Computer system is a relatively closed system. It does not disturb from outside the system. Chemical reaction in the sealed tube also is an example of closed system.

4. **Natural and Artificial Systems:**
- Natural systems exist and also bound in nature like river, mountains etc., and while artificial systems are manufactured (man-made) such as dams, canals, roads etc.
- For Example, Solar System.

5. **Man-machine System or Man-made Information System:**
- Generally most of the artificial systems are man-machine systems.
- For example, a motor bike is a machine system but a bike cannot work without a person.
- The systems developed and engineered by human are a man-made system.
- Information System (IS) is man-machine systems relatively closed and deterministic system.
- Information system may be defined as, "a set of devices, procedures, and operating systems designed around user-based criteria to produce information and communicate it to the user for planning, control and performance."
- Man-made information systems are divided into three types:

(i) **Formal Information System:**
- A formal information system is represented by Organisation chart. The chart is a map of position and their authority relationship. Indicated by boxes and connected by straight lines.
- It gives a representation of the different parts of system and flow of information in the form of memos, instructions, etc., from top level to lower levels of management.

Types:
- There are three types of formal information system:
  1. Strategic Information System
  2. Managerial Information System
  3. Operational Information System

1. **Strategic Information System:**
- Strategic information system relates to long range planning policies. These are the direct interest of the upper management. Polices are the generalisation that specify what an organization is going to do. For Example, financial investment in a company, human resources etc.

2. **Managerial Information System:**
- **Managerial information system** relates to short and intermediate range planning policies. This is done by the direct interest of middle management and departmental heads for implementation and control.

3. **Informal Information System:**
- **Operational information system** relates to short term polices. This deals with day-to-day activities like daily employee absence sheets, purchase orders, current stock available for sale.
- This is an employee based system designed to meet personnel and vocational needs and help work related problem.
- Informal information system is related with what is happening practically rather than what is shown on paper.

(ii) **Computer Based Information System:**
- A Computer Based Information System (CBIS) is an information system that uses computer technology to perform some or all of its intended tasks. Today most information systems are computerized, although not all of them are. For this reason the term "Information System" is typically used synonymously with "Computer Based Information System."
- This system is directly dependent on the computer for managing business applications. For example, Automatic Library System, Railway Reservation System, Banking System, etc.
- Computer based information systems are faster, more accurate, more neat and attractive. It is possible to perform different operations easily. Security of data possible in this system.
- CBIS can be categories as follows:

(i) **A Decision Support System (DSS)**
- DSS systems make use of analytical planning modules (operation research model).
- DSS mostly used for assisting top-level management in decision making. Better decisions are taken by using DSS. DSS reduces clerical work and overtime.
- DSS also saves cost and time. It consists of decision making with support of other lower level systems (MIS).
- DSS uses two types of data collected from environment of the Organisation:
  1. Internal data.
  2. External data.
- Internal data mostly used for studying the trends while external data is mostly used for understanding the environment.

Benefits of DSS:

1. DSS improving personal efficiency.
2. DSS improving problem solving.
3. DSS increasing organisational control.

Examples:

- Bank loan management systems
- Financial planning systems

(ii) Transaction Processing System (TPS)

- This is the most fundamental computer based system in an organization. This relates to the processing of business transactions.
- A transaction processing system can be defined as a system that captures, classifies, stores, maintains, updates and retrieves transaction data for record keeping and input to the other types of CBIS.
- Transaction Processing System is targeted at improving the routine business activities. A transaction is any event or activity that affects the whole organization.
- Examples of transaction processing systems are:
  - Airline booking systems – flights booking management
  - Point of Sale Systems – records daily sales
  - Payroll systems – processing employee's salary, loans management, etc.
  - Stock Control systems – keeping track of inventory levels

(iii) Management Information System (MIS)

- A MIS is a system that provides historical information, information on the current status. It is a communication process in which data are recorded and processed for further operational uses.
- An MIS is a set of Computer Based System and procedures implemented to help managers in their essential job of decision making. The actual process will involve the collection, organization, distribution and storage of organization wide information for managerial analysis and control.
- MIS is made up of three components:
  - (a) **Management**: Emphasizing the ultimate use of such information system for decision making.
  - (b) **Information**: Information highlighting on processed data rather than the raw data and in the context in which managers and other end users use it.
  - (c) **System**: Highlighting a fair degree of integration.

- The MIS system investigates the input with routine algorithms i.e. aggregate, compare and summarizes the results to produced reports that tactical managers use to monitor, control and predict future performance.
- For example, input from a point of sale system can be used to analyze trends of products that are performing well and those that are not performing well. This information can be used to make future inventory orders i.e. increasing orders for well-performing products and reduce the orders of products that are not performing well.
- Examples of Management Information Systems are:
  - **Human Resource Management Systems:** These systems combine a number of systems and processes to ensure the easy management of human resources such as overall welfare of the employees, staff turnover, etc.
  - **Sales Management Systems:** Sales management is the process of co-ordinating the salespeople in the organization to work towards a common goal.
  - **Budgeting Systems:** These systems give an overview of how much money is spent within the organization for the short and long terms.

(iv) Office Automation System (OAS)

- Office Automation Systems are among the newest and most rapidly expanding computer based information systems.
- This system is the CBIS that combines various technologies to reduce the manual labor needed to operate an office efficiently; used at all levels of an organizations.

Other Types of Systems:

1. Expert System (ES)

- This is also called knowledge-based system. It is a set of computer programs that perform a task at the level of a human expert.
- ES operates with few rules. Effectiveness is a major goal of these types of systems. Human beings are experts in specific areas.
- ES are more flexible than other systems. ES increases output and productivity of the system.
- ES gives effective manipulation of large knowledge based system. The output is selected with the opinion of many experts.

Components:

- Expert system consists of following components:
  1. User interface.
  2. Explanation facility.

3. Knowledge acquisition.
4. Knowledge base, facts rules.
5. Knowledge refining system.

Advantages of ES:

1. ES improves quality by providing consistent advice and by making reduction in error rate.
2. ES increase output as an ES works faster than human being.
3. An ES can perform as a single human expert in many problem situations.
4. ES helps to preserve and reproduce knowledge of experts.

2. Executive Information System (EIS)

- This system also called an **Executive Support System (ESS)**. It is made especially for top managers that specifically supports strategic decision making.
- EIS is structured tracking system. It provides rapid access to timely information and direct access to management reports.
- EIS contains extensive graphics capabilities. It serves the information needs of top executives. EIS gives quick and easy access to detailed information.

Advantages of EIS:

1. EIS is easy for upper level executives.
2. EIS provides timely delivery of company summary information.
3. EIS improves to tracking information.
4. EIS filters data for management.
5. EIS offers efficiency to decision making.

1.7 SYSTEM COMPONENTS

- A system is a group of interrelated components working together toward a common goal or objective by accepting inputs and producing outputs in an organized transformation process.
- A system component is a process, program, utility, or another part of a computer's operating system that helps to manage different areas of the computer.
- A system has following system components:
  1. Components
  2. Inter-related components
  3. Boundary
  4. A purpose
  5. An environment
  6. Interfaces

7. Constraints
8. Input
9. Output

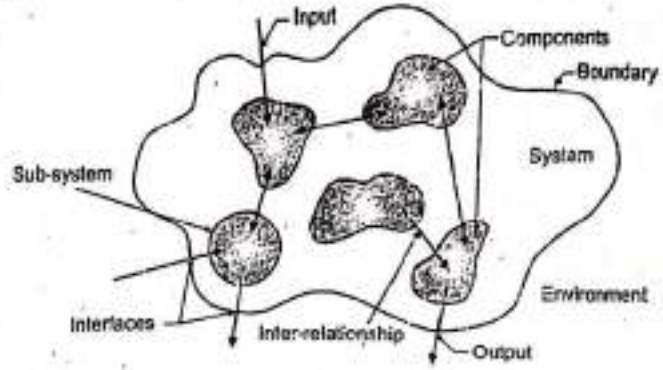


Fig. 1.8: Components of a System

- Various system components in Fig. 1.8 are described below:

1. **Components/Sub-system:** A component is either a complex part or an a of parts, also called a sub-system. For example, in case of an Automobile repair or upgrade the system by changing individual components without to make changes the entire system.
2. **Inter-relationship:** The components are inter-related i.e., the function o somehow tied to the function of the others. For example, in the Store sys work of one component, such as producing a daily report of customer orde not progress successfully until the work of another component is finishe as sorting customer orders by date of receipt.
3. **Boundary:** A system has a boundary within all of its components are co and which establishes the limits of a system, separating it from other syst of the components work together to achieve overall purpose. Purpose me overall objective/goal of a system.
4. **Environment:** A system operates within an environment; everything outs system's boundary. The environment surrounds the system, both affectin being affected by it. For example, the environment of a University i prospective students, foundations, funding agencies and the new media. the system interacts with its environment. A university interacts with pros students by having open houses and recruiting from local high schools.

5. **Interface:** The point at which the system meets its environment are called interface.
6. **Constraints:** A system must face constraints in its functioning because there are limits to what it can do and how it can achieve its purpose within its environment. Some of these constraints are imposed inside the system (For example, a limited number of staff available). Others are imposed by the environment (For example, due to regulations).
7. **Input and Output:** A system interacts with the environment by means of input and output. Input is anything entering the system from the environment while output is anything leaving the system crossing the boundary to the environment. For example, an Electrical utility takes on input from the environment in the form of raw materials (coal, oil, water power, etc), requests for electricity from customers. It provides for output to the environment in the form of electricity.

- Fig. 1.9 shows an example a Fast-food Restaurant System.

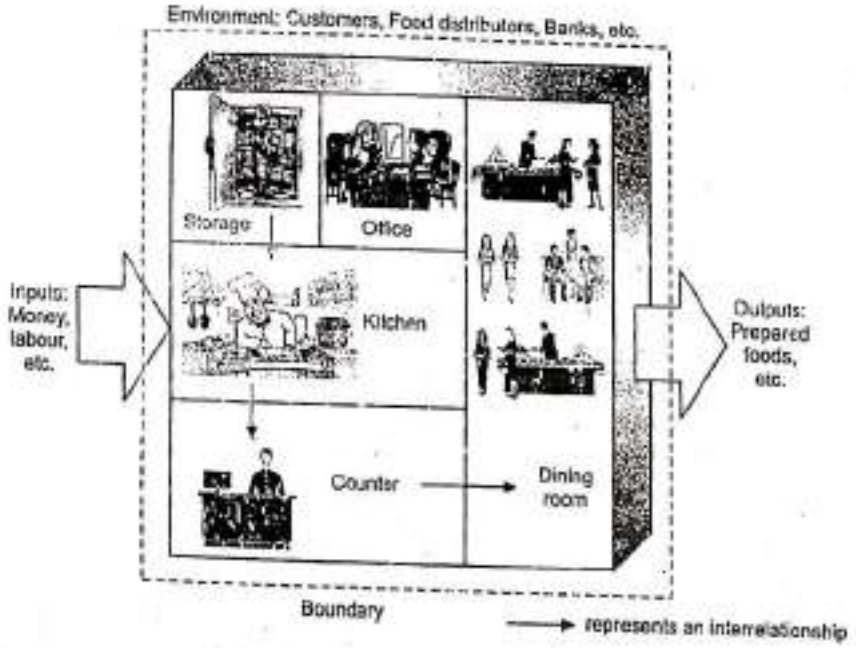


Fig. 1.9: A Fast-food Restaurant as a System

Summary

- System analysis and design is a problem-solving procedure for examining an information system and improving it.
- An Information System (IS) is a set of inter-related components that collect, store, process, help analyze, support decision making and perform control functions in an organization.
- The purpose of systems analysis is to gather data, analyze the data, and write a report and the purpose of systems design is to do a preliminary design and then a detail design, and write a report.
- A system is defined as, a collection of related components that interact to perform a task in order to accomplish a goal.
- Input in a system is whatever it takes from its environment in order to fulfill its purpose.
- Component in a system is an irreducible part or aggregation of parts that make up a system and also known as a sub-system.
- Inter-related components in a system means dependence of one sub-system on one or more sub-systems.
- Boundary of a system is the line that marks the inside and outside of a system and that sets off the system from its environment.
- Purpose component of system means the overall goal or function of a system.
- Environment in a system represents everything external to a system that interacts with the system.
- Interface in a system represents point of contact where a system meets its environment or where sub-systems meet each other.
- In a system, constraint is the limit to what a system can accomplish.
- In a system, output is whatever a system returns to its environment in order to fulfil its purpose.
- DSS is made up of three components i.e., Decision (emphasizes decision making in problem situation, not in information processing, retrieval or reporting), Support (requires computer aided decision situations with enough "structure" to permit computer support) and System (accentuates the integrated nature of problem solving, suggesting a combined man, machine and decision environment).
- Expert System (ES) is a computer-based information system that uses its knowledge about a specific complex application area to act as an expert consultant to users. The ES consists of a knowledge base and software modules that perform inferences on the knowledge, and communicate answers to a user's questions.

- OASs are the newest and most rapidly expanding computer based information systems. Office automation refers to the varied computer machinery and software used to digitally create, collect, store, manipulate, and relay office information needed for accomplishing basic tasks. Raw data storage, electronic transfer, and the management of electronic business information comprise the basic activities of an Office Automation System (OAS).
- Open system is a type of system that interacts freely with its environment, taking input and returning output.
- Closed system is a type of system that is cut off from its environment and does not interact with it.
- TPS keeps track of the transactions needed to conduct business and used in decision making.
- MIS derives data from all departments of an organization and produces summary, exception, periodic and on-demand reports of the organization's performance.
- Deterministic system is system which acts in a predictable manner where stepwise execution and the output are already known.
- Probabilistic system is a system which acts in an unpredictable manner and where the outcome is not predictable.
- These days information system relies on computer for the storage, retrieval of data. Computers are used to make business application. These days system analysis heavily relies on computer to solve the business problem for this Computer Based Information System (CBIS) are used.

Check Your Understanding

1. What is a Software?
   (a) Software is set of programs
   (b) Software is documentation and configuration of data
   (c) Software is set of programs, documentation & configuration of data
   (d) None of the above
2. The condition outside a system is called _____.
   (a) Interface (b) Environment
   (c) Boundary (d) All of the above
3. The conceptual system also known as _____.
   (a) Abstract system (b) Physical system
   (c) Open system (d) Closed system

4. Which of the following system keeps track of the transactions needed to co business and used in decision making?
   (a) MIS (b) OAS
   (c) TPS (d) DSS
5. A system is collection of components, that work together to achieve a _____
   (a) Speed (b) Collaboration
   (c) Common goal (d) Limit
6. Which is not the characteristic of a system?
   (a) Structure (b) Central objective
   (c) dependence (d) Interdependence
7. Which of the following is not the element of system?
   (a) Control (b) Input
   (c) Environment (d) Risk
8. Physical systems are _____ entities that may static or dynamic in nature.
   (a) Tangible (b) Intangible
   (c) Weak (d) Strong
9. Interconnection and interaction between subsystems are known as _____.
   (a) Feedback (b) Interfaces
   (c) Environment (d) Boundaries
10. Which of the following is not the level of processes?
    (a) DSS (b) TPS
    (c) MIS (d) SRS
11. The system which is directly interacting with the environment is kno _____.
    (a) Abstract system (b) Open system
    (c) Closed system (d) Transaction processing system.

ANSWER KEY

1. (c)	2. (b)	3. (a)	4. (c)	5. (c
6. (c)	7. (d)	8. (a)	9. (b)	10. (
11. (b)				

Practice Questions

Q.I: Answer the following Questions in short.

1. What is the system? Explain with example.
2. Define sub-system?
3. What are the three basic components of any system?
4. What are the elements of any system?
5. What is the informal information system?
6. Define interface of a system with example.
7. Define physical and abstract systems.
8. What is the deterministic system?

Q.II: Answer the following Questions.

1. Define system? Describe types of system.
2. Explain characteristics of a system?
3. Differentiate between physical and abstract system.
4. What are open and closed systems?
5. What are the characteristics of open system?
6. Describe the MIS.

Q.III: Define the following terms.

1. Interface
2. Feedback
3. Control
4. Boundary
5. CBIS
6. OAS
7. Formal System
8. Expert System
9. Input and Output
10. Natural System

Previous Exams Questions

BBA (CA)

Summer 2018

1. Define system and its elements.

Ans. Refer to Sections 1.1.2 and 1.1.3. [2M]

2. Distinguish between TPS and DSS.

Ans. Refer to Sections 1.3 and 1.4.3. [2M]

3. Write short note: System characteristics.

Ans. Refer to Section 1.2. [4M]

Winter 2018

1. Define Interface concept of system with example.

Ans. Refer to Section 1.5 [2M]

2. Explain the types of formal information system.

Ans. Refer to Section 1.6 [4M]

3. Expert System

Ans. Refer to Section 1.3 [4M]

Summer 2019

1. Define open and closed system.

Ans. Refer to Section 1.6 [2M]

2. Write Short note: Element of System

Ans. Refer to Section 1.5 [4M]

BCA (Science)

Summer 2018

1. Which of the following is not the element of the system? [1M]
   (a) Control (b) Feedback
   (c) Risk (d) Environment

Ans. Refer to section 1.3

2. Robotics comes under which category? [1M]
   (a) System software (b) Artificial intelligence
   (c) Business software (d) Net sourcing

Ans. Refer to Section 1.5

3. Explain characteristics of system.

Ans. Refer to Section 1.2 [4M]

Winter 2018

1. It is the super system in which any organisation operates: [1M]
   (a) Environment
   (b) Boundaries

(c) Interfaces
(d) Subsystem

Ans. Refer to Section 1.4 **[3M]**

2. Explain any three characteristics of a system. **[3M]**

Ans. Refer to Section 1.2

Summer 2019

1. ..... is a computer based information system that uses its knowledge about specific complex application area to act as an expert consultant to users. **[1M]**

(a) Expert system
(b) Knowledge system
(c) Control system
(d) None of the above

Ans. Refer to Section 1.5 **[1M]**

2. Define open system. **[1M]**

Ans. Refer to Section 1.5

❖❖❖

2... Introduction to Software Engineering

Objectives...

- To understand meanings of Software and Software Engineering.
- To know the need of Software and software application domains.
- To learn about Mc Call's Quality factors.
- To study Software Process and its Concepts.

2.1 INTRODUCTION

- Software engineering is a new technological disciple which is distinct from, but ba on the foundations of computer science, management science, econom communication skills, and the engineering approach to problem solving.
- The term is made of two words, **Software** and **Engineering** as explained below:

Software: Software is more than just a program code. A program is executable code, which serves some computational purpo Software is considered to be collection of executable programmi code, associated libraries and documentations. Software wh made for a specific requirement is called software product. software product has to perform certain specific function/t required by users (customers).

Engineering: Engineering is all about developing products, using well-defin scientific principles and methods.

- Software engineering requires both technical skills (Computers, Operating Syst Database, DBMS, Computer Networks, Information Systems (ISs), Sy development tools, etc.) and managerial control (Project management,

management, Change management, Research management etc.) because of it being a labor-intensive activity.

- Software engineering activities occur within an organizational context and a high degree of communication is required among customers, managers, software engineering, hardware engineers and other technologists.
- This chapter focuses on nature of software, software engineering, its product and process and quality factors.

2.2 DEFINITION OF SOFTWARE (BCA: S-18; BBA(CA): W-18, S-19)

- Software is defined as, "a set of instructions to acquire inputs and to manipulate (process) them to produce the desired output in terms of functions and performance as determined by the user of the software."

OR

- A software is, "a set of instructions, (computer programs) that when executed provide desired output, performance and function".

OR

- Software is defined as, "a data structure that enables the programmer to adequately manipulate information".
- Software is logical rather than a physical system element.

Importance of Software:

- Today computer software has become a driving force, because of following reasons:
  1. It serves as the basis for modern scientific investigation and engineering problem solving.
  2. It is embedded in all kinds of systems like medical, telecommunications, military, industrial processes, entertainment etc.
  3. It is engine that drives business decision-making.

2.3 CHARACTERISTICS OF SOFTWARE (BBA(CA): S-19)

- There are various characteristics of software as explained below:

1. **Software is Developed or Engineered; Software is not manufactured in the Classical Sense:**

- There are some similarities between software development and hardware manufacturing that is high quality is achieved through good design, but the manufacturing phase for hardware can introduce quality problems that are non-existence (or easily corrected) for software.
- Software costs are condensing in engineering. This means that software projects cannot be managed as if they were manufacturing projects.

2. **Software does not "Wear Out":** (BBA (CA): W-18)

- Fig. 2.1 shows failure rate as a function of time for hardware.
- The relationship often called the 'bathtub curve' indicates that hardware exhibits relatively high failure rates early in its life, defects are corrected and the failure rate drops to a steady state level for some period of time.

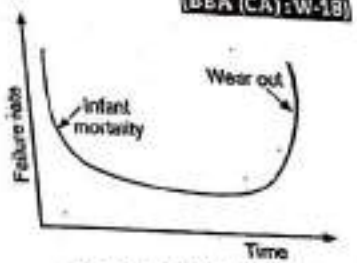


Fig. 2.1: Bathtub Curve

- As time goes, the failure rate increase again as hardware components suffer from the cumulative effects of dust, vibrations, abuse, extreme temperature and many environment components. All these things state that the hardware begins to wear out.
- Software is not susceptible to the environmental components due to this hardware to wear out. Ideally, the failure rate curve for software should take the form of the curve shown in Fig. 2.2.
- Undiscovered defects will cause high failure rates in the life of program. During its life, software will undergo maintenance (or change).

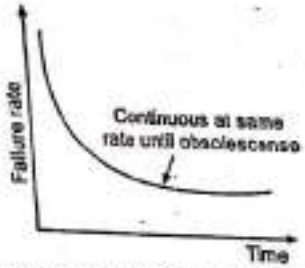


Fig. 2.2: Idealized Software Failure Curve

- As changes are made, it is likely that some new defects will be recognized, causing the failure rate curve to spike as shown in Fig. 2.3.

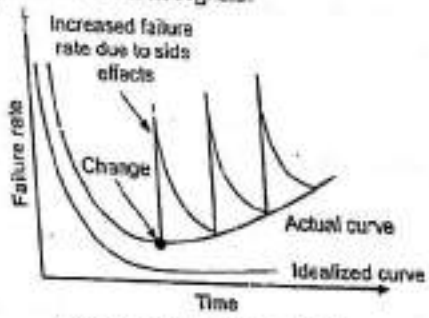


Fig. 2.3: Actual curve of Software

3. **Most Software is Custom Built, rather than being assembled from existi components:**

- As an engineering discipline evolves, a collection of standard design component created.

- In the hardware design, engineer draw a software schematic of the digital circuitry, does some basic analysis to assure proper functioning, goes through a catalog of a digital components. In the hardware world, component reuse is a natural part of the engineering process.
- On the other hand, software designer that has only began to be achieved on a broad scale. A software component should be designed and implemented so that it can be reused in many different programs.
- Various versions of the software are possible. New modifications or updations are done in existing software to form a new version of software.

2.4 SOFTWARE APPLICATION DOMAINS

- A domain defines a set of common requirements terminology, functionality for any software program constructed to solve a problem in the area of computer programming environment.
- An application domain is a mechanism used to isolate executed software applications from one another so that they do not affect each other's functionality.
- There are seven main categories of software domains as explained below:

1. **System Software:**

- System software is a collection of programs written to service other programs.
- Examples: Compilers, File Management Utilities, Device Drivers, Editors and Operating Systems.
- These programs are heavily interacting with hardware components.

2. **Application Software:**

- Application software is a stand-alone program. It solves a specific business requirement. It uses technical data which facilitates business operations.
- Examples: Real-time manufacturing Process control, Payroll system etc.

3. **Engineering/Scientific Software:**

- Engineering/scientific software uses various number crunching algorithms. Applications range from astronomy to volcanology, from automotive stress analysis to space shuttle orbital dynamics etc.
- But new applications may not base on the conventional numerical algorithms.
- Examples: System simulation, Computer Aided Design (CAD), etc.

4. **Embedded Software:**

- Embedded software resides within a system or product. It is used to implement and control features and functions for the end-user for the system itself.
- Embedded software can perform limited and esoteric functions.

- Examples: Key pad control for refrigerator, Software in mobile phone, Softwa Anti Lock Braking in car, Software in microwave oven to control the cooking proce

5. **Product Line Software:**

- Product Line software is developed/designed to provide a specific capability for u by different customers.
- It can focus on limited and esoteric market place like inventory control products.
- Examples: Spreadsheets, Software for Entertainment, Business finan applications, Computer graphics, Multimedia, DBMS etc.

6. **Web Application:**

- Web application is also called 'Web Apps'.
- This network based software having wide array of applications.
- It can be more than a set of linked hypertext files which presents information u text and limited graphics.
- Web Apps are evolving into sophisticated computing environment that not provide stand-alone features, computing functions, and content to the end user also are integrated with corporate database and business applications.
- Examples: Web applications include online forms, shopping carts, word proces spreadsheets, video and photo editing, file conversion, file scanning, and e programs such as Gmail, Yahoo and AOL.

7. **Artificial Intelligence (AI) Software:**

- AI software uses the non-numerical algorithms to solve complex problems.
- **Examples:** The applications includes in AI are robotics, expert systems, pat recognition, artificial neural networks, games playing, theorem proving etc.

2.5 DEFINITION OF SOFTWARE ENGINEERING (BBA(CA)

- Computer software is a product that design and built by software engineers, develop product by using software engineering approach.
- Software has becomes the key element in the evolution of computer based system product.
- Software engineering is concerned with development and maintenanc technological products, problem solving techniques which are followed i engineering disciplines like project planning, project management, system ana design and outgoing maintenance activities within cost estimates.
- A primary goal for software engineering is to improve the quality of software pr and to increase the productivity and job satisfaction of software engineers.

Definition:

- Software engineering is, "the technological and managerial discipline concerned with systematic production and maintenance of software products which all developed and modified on time within the cost estimates".

OR

- Software engineering defined as, "the application of a systematic, disciplined quantifiable approach to the development, operation and maintenance of software, i.e., the application of engineering to software".

2.5.1 Layered Technology of Software Engineering (BCA: S-19)

- Software engineering is a cut growth of hardware and system engineering. Software engineering can be viewed as a layered technology. It consists of a set of key elements (layers) i.e. A quality focus, Process, Methods, Tools as shown in Fig. 2.4.

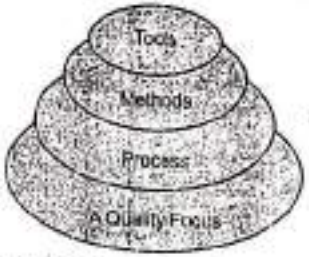


Fig. 2.4: Software Layered Technology

These elements of Software Layered Technology are explained below:

1. **A Quality Process Layer:** An engineering approach must have a focus on quality which provides a continuous process improvement culture.
   The software product should fulfill the customer quality requirements (i.e efficiency, reliability, etc), developer quality requirements (maintainability, reusability, etc), users (usability, efficiency etc).
2. **Process Layer:** Foundation for Software Engineering is the Process Layer. Software Engineering process holds all the technology layers together and enables the timely development of computer software. It forms the bare for management control of software project.
3. **Methods Layer:** It provides technical knowledge for developing software. The method layer covers a broad array of tasks which include requirements analysis, design, coding, testing and maintenance phases of the software development.
4. **Tools Layer:** It provides automated or semi-automated support for the Process and Method layer. These tools are used to bring automation in software development process Sometimes, tools are integrated in such a way that other tools can use information created by one tool i.e., Computer-Aided Software Engineering (CASE).

2.6 NEED FOR SOFTWARE ENGINEERING

- The need of software engineering arises because of higher rate of change in user requirements and environment on which the software is working.
- A major challenge for software engineering today is to improve the software programming process as modern software life cycle has been changing very dramatically, wherein the code re-usability, reliability and maintainability are the key features.
- Following points describes need of software engineering:
  1. **For efficient use of Resources:** As software development is expensive so proper measures are required, so that the resources are used efficiently.
  2. **For dynamic (Changing) nature of Software:** The always growing and adapting nature of software hugely depends upon the environment in which user works. If the nature of software is always changing, new enhancements need to be done in the existing one. This is where software engineering plays an important role.
  3. **For reduction of Complexity:** Today the complexity of software project is increased. The problem of software complexity can be solved easily with the help of software engineering.
  4. **For Quality Management:** Better process of software development provides better and quality software product.
  5. **For Reduction of Cost and Time:** Cost and time considerations are another factor, which arises the need for software engineering.
  6. **To Maintain Effectiveness of Software:** Effectiveness comes if anything has made according to the standards. Software standards are the big focus of companies to make it more effective. So software becomes more effective in performance with the help of software engineering.
  7. **For Production Reliable Software:** Software should be reliable that means if the delivered software should work as per user's expectations without any errors.
  8. **For Production Large Software:** It is easier to build a wall than to a house or building, likewise, as the size of software become large engineering has to step to give it a scientific process.
  9. **For Improve Scalability:** If the software process were not based on scientific and engineering concepts, it would be easier to recreate new software than to scale an existing one.
  10. **To Increase Productivity:** Software requires a lot of hard work and software engineers are highly paid professionals. A lots of man force is requires to develop software with millions of codes. But in software engineering, programmers plan everything and reduce all those things that are not required which results in reduction of cost.
- In short, software engineering is needed for develop complex and critical software systems in a timely manner, with high quality and low cost.

(ii) **Reusability:** Extent to which parts of a software system can be reused in other applications. Reusability means if a significant portion of one product can be reused, maybe with minor modifications, in another product.

(iii) **Interoperability:** The effort required to couple one system to another.

- Interoperability means whether or not the output of one system is acceptable as input to another system, it is likely that the two systems run on different computers interconnected by a network.
- An example of interoperability is the ability to roam from one cellular phone network in one country to another cellular network in another country.

2.7.1 Quality Criteria

- A quality criteria is an attribute of a quality factor that is related to software development. For example, modularity is an attribute of the architecture of a software system.

List of Quality Criteria:

1. **Access to Audit:** Ease with which the software and data can be checked for compliance with standards.
2. **Accuracy:** Precisions of computations and output.
3. **Access Control:** Provisions for control and protection of the software
4. **Completeness:** Degree to which full implementation of required functionalities have been achieved.
5. **Communicativeness:** Ease with which the inputs and outputs can be integrated.
6. **Conciseness:** Compactness of the source code, in terms of lines of code.
7. **Consistency:** Use of uniform design and implementation techniques.
8. **Data commonality:** Use of standard data representation.
9. **Error tolerance:** Degree to which continuity of operation is confirmed under adverse conditions.
10. **Execution efficiency:** Run time efficiency of the software.
11. **Expandability:** Degree to which storage requirements or software functions can be expanded.
12. **Hardware independence:** Degree to which a software is dependent on the underlying hardware.
13. **Modularity:** Provision of highly independent modules.
14. **Operability:** Ease of operation of the software.

15. **Simplicity:** Ease with which the software can be understood.
16. **Software efficiency:** Run time storage requirements of the software.
17. **Traceability:** Ability to link software components to requirements.
18. **Training:** Ease with which new users can use the system.

2.8 THE SOFTWARE PROCESS

- Software is developed/engineered effectively and efficiently with the help of processes (activities). In general a process is defined as, "a series of steps involving activities and resources which produces the desired/expected output."
- Software process is defined as, "the related set of activities and processes that are involved in developing and evolving a software system."

OR

- Software development process is defined as, "a collection of procedures to develop the software product according to certain goals or standards."
- Generally, the following points are noted about the software process.
  1. It includes guidelines which explain the objectives of each activity.
  2. It includes various technical and managerial issues that are required to develop the software.
  3. It uses resources subject to given constraints and produces intermediate and final products.
  4. It is carried out with an entry and exit criteria that help in monitoring the beginning and completion (ending) of the activity.
  5. It is regarded as more than just a procedure, tools and techniques which are collectively used in a structured manner to produce a product.

Characteristic of Software Process:

- The characteristics of software processes are listed below:
  1. **Understandability:** The extent to which the process is explicitly defined and the ease, simple with which the process definition is understood.
  2. **Visibility:** Whether the software process activities culminate in clear result or output so that the progress of the process is visible externally.
  3. **Supportability:** The extent to which CASE tools can support the process activities.
  4. **Reliability:** The manner in which the software process is designed such that errors in the process are avoided (trapped) before they result in errors in the product.
  5. **Rapidity:** The speed with which the complete software can be delivered with given specification.

6. **Acceptability:** The extent to which the defined software process is acceptable and usable by the software engineers/developers for producing the software product.
7. **Robustness:** Whether the software process can continue in spite of unexpected problems.
8. **Maintainability:** Whether the process can evolve to reflect the changing organizational requirements or identify improvements in the process.

2.8.1 Software Process Model

- A process model provides a basis for controlling various activities required to develop and maintain the software.
- Fig. 2.6 shows the basic generic software process model.

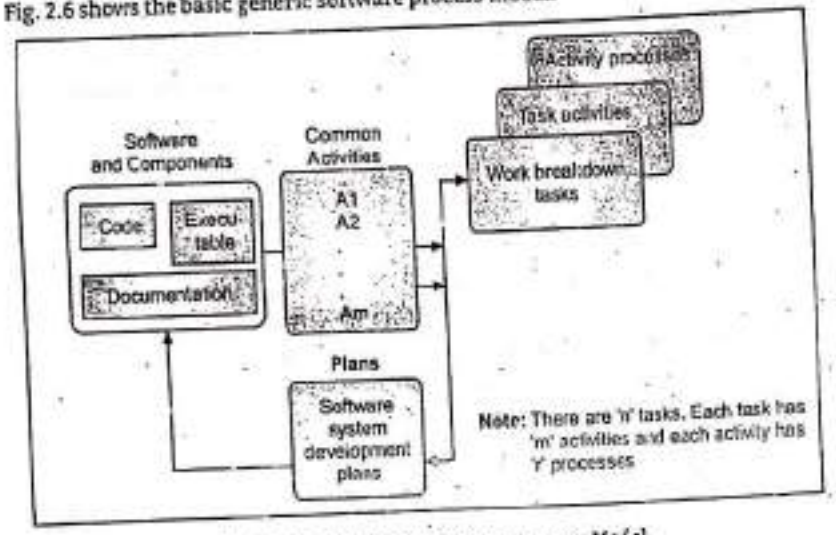


Fig. 2.6: Basic Generic Software Process Model

- A software process model is defined as, "a strategy (software engineering paradigm), comprising process, methods and tools layers as well as the general phases for developing the software."
- A process model for software engineering depends on the nature and application of the software project.

2.8.2 Software Process Framework Activities

- A process framework provides a foundation for a complete software engineering process by identifying small number of activities.

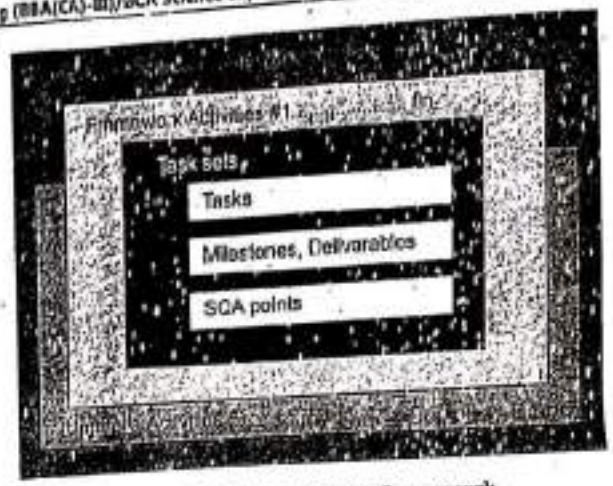


Fig. 2.7: Software Process Framework

- These activities are known as framework activities. These are applicable software projects regardless of their type and complexity.

A generic process framework includes following five activities:

1. **Communication:**
- Before starting any technical work, it is very important to communicate collaborate with the customer/user.
- The main goal is to understand the objective/goal of the customer behin development of a system i.e. why he/she wants a system to be developed?

2. **Planning:**
- If a map exists any complicated journey becomes easy. The planning creates a which helps to guide the developers/engineers in the development of system.
- The map is called as software project plan. Map define technical work, risk, reso used, work schedule and so on.

3. **Modeling:**
- If we create sketch of thing, then we can simply understand the big picture system and also other characteristics requirement of the system. We can refi sketch.
- A software engineering does the same thing by creating models to better under So model of working condition must be developed. It is a design process.
- In other words, modeling encompasses creation of model, which allows the so engineer and the user to understand software requirement and designs to a those requirements.

4. **Construction:**

- It combines code generation with testing to uncover errors in the code.
- It finds the errors and makes the system workable.

5. **Deployment:**

- The delivery of working system for evaluation and feedback is called Deployment.
- It implies that the final product i.e. the software is delivered to the customer/user.
- Deployment is done in customers place to find whether all the system activities are working fine or not.

Umbrella Activities: **(BCA: S-18)**

- In addition to framework activities, process framework also comprises a set of activities known as **Umbrella Activities** which are used throughout the software process.
- These umbrella activities are explained below:

1. **Software Project Tracking and Control:** This activity monitors the actual process so that the management can take necessary steps if the software project move away from the plans laid down. It involves tracking procedures and reviews to check whether the software project is according to user requirements. It contains a documented plan used as basis for tracking the software activities and for revising the plans.
2. **Formal Technical Reviews(FTR):** This activity consider the code, products and documents of software engineering practices to detect and remove errors.
3. **Software Quality Assurance (SQA):** This activity assures that the software is according to the user requirements. In addition, it is used to evaluate the processes of developing and maintaining the quality of the software.
4. **Reusability Management:** This activity determines the criteria for product's re-use and establishes mechanisms to achieve re-usable components.
5. **Software Configuration Management (SCM):** SCM manages the changes made in the software processes throughout the life cycle of the software project. It also controls changes made to the configuration and maintains the integrity of the software.
6. **Risk Management:** This activity identifies, analyses, evaluates and eliminates the possibility of unfavorable deviations from the expected results in a systematic manner and then develops strategies to manage them.

2.9 SOFTWARE PRODUCT AND PROCESS

Software Product:

- The final software that is delivered to the customer with documentation is called the software product. It is the outcome of the entire software development process.
- It describes how to install and use the system. It may include source code, data, user guides, reference manuals, installation manuals, specification documentation, other documentation, etc.
- The software product does not have any information regarding the software process, like how it was scheduled, how many people worked on it, how the work was divided, etc. It only consists of the final application that fulfills the user's requirements.
- In certain cases, software products may be part of system products where hardware, as well as software, is delivered to a customer.
- Types of software products: Software products fall into two broad categories:
  1. **Generic products:**

     Generic products are the stand-alone systems that are developed by a production unit and sold on the open market to any customer who is able to buy them.
  2. **Customized Products:**

     Customized products are the systems that are commissioned by a particular customer. Some contractor develops the software for that customer.
- The software process includes all the activities that are performed to the form the final Software product, like the requirement analysis, designing of the software, coding, testing, documentation, Maintenance, etc. Hence, the software product can also be defined as the collection of all the activities that as a result leads to the formation of the software product.
- The Software Product may not contain details about the software process, but the software process has every detail about the final product from the very initial phase itself that how the software would be like.
- The two terms, the software product and the software process are related to each other. An efficient process is very important to produce a good quality software product. If the software development process is weak, then the final product will definitely suffer.
- However, the software product is more dependent upon the software process. For example, If we have a software product, then it would certainly have its history which is the software process. But, the case is not the same with Software process. An ongoing software process doesn't need to lead to a final product. There are chances that some problems may occur in the projects development phase and it may be cancelled.

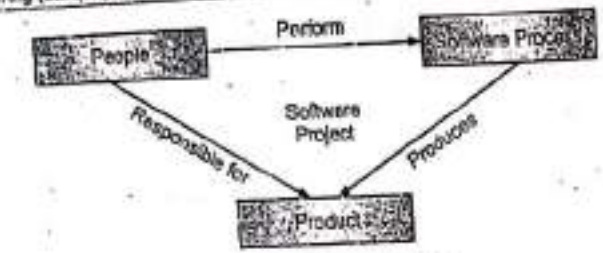


Fig. 2.8: Software Project

- Software process can be used in the development of many software projects; each of which can produce one or more software products. Software project begins with requirements and ends with the fulfilment of requiremnets. The interrelationship among these three entities(process, project and product) is shown in following Fig 2.9.

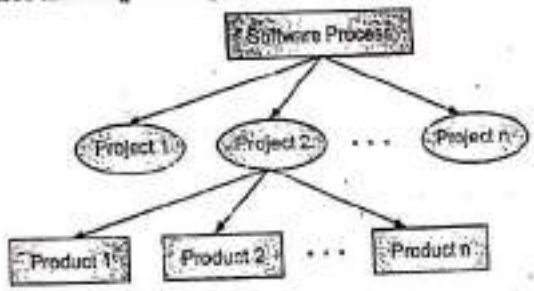


Fig. 2.9: Process, Projects and Products

2.10 SOFTWARE ENGINEERING PRACTICE

- The framework activities and umbrella activities establish a skeleton architecture for software engineering work.
- Software Engineering Practice is a collection of concepts, principles, methods, and tools that a software engineer calls upon on a daily basis.
- This practice allows managers to manage software projects and software engineers to build computer programs.
- It provides necessary technical and management how to getting the job done.
- Practice transforms a disorganized, unfocused approach into something that is more organized, more real, efficient and more likely to achieve success.
- Principles are important because of they help us to establish a mind set for solid software engineering practice.
- This section discusses the essence of practice and general (core) principles.

2.10.1 The Essence of Practice

(BCA: S

- Essence is a standard that defines the smallest set of concepts that are common to software projects.
- Practice is a broad array of concepts, principles, methods, and tools that you m consider as software is planned and developed.
- George Polya outlined the essence of problem solving and consequently, the esse of software engineering practice. He suggested the following things:

1. Understand the Problem (Communication and Analysis)

- It is always better to understand the problems or difficulties arrived in the sys development.
- For this thing, customers are the important entities. We can find the problems w they are facing in the available or manual system.
- Which data, function and features are required to solve the problem?
- Is it possible to divide the full system into subsystem that may be easie understand?
- Can we represent the problem graphically? i.e., it is possible to develop a GUI b system has to be considered by the developer.

2. Plan a Solution (Planning, modeling and software design)

- It is taking case of design part. Consider that is there any such system available a system design is ready. If so, revise it with new specification else design a system. If an existing system is available whether it is effectively implemented.

3. Carry out the Plan (Construction; code generation)

- The coding is done for the system to be developed.
- Check whether the source-code is traceable to the design model or not.
- Check the working components i.e., sub–modules.

4. Examine the Result (testing and quality assurance)

- Check the solution with respect to perfection and accuracy.
- Test all the components and sub-systems.
- Check the result with the specifications given by customer.

2.10.2 General Principles

(BCA: S

- David Hooker has proposed seven general (core) principles which focus on softw engineering practices as whole. These principles are as follows:

Principle# 1: Remember the reason that the software exists

- A software system exists for one reason to provide value to its users.

- The hardware platforms or development process, are should think for the real value to the system.

Principle# 2: Keep it simple

- The design should be simple so that system must be easily understood and easily to maintained.
- Simple design does not mean that quick and dirty design.

Principle# 3: Maintain the Vision of the project

- A clear vision is essential for the success of a software project. And it should be maintained.
- We should have an architect who can hold the vision and enforce compliance helps ensure a better successful software project.

Principle# 4: What you produce, other will consume

- Always design, specify and implement knowing that someone else will later have to understand and modify what you did.
- Someone may debug your code without asking you about the details of the code.

Principle# 5: Be Open to the future

- A system which has long lifetime has more value.
- The lifetime of software is measured in months rather than years.
- All possible solutions must be available in the system of there is a change required in a system.

Principle# 6: Plan Ahead for software Reuse

- Reusability saves time and effort.
- The reuse of code and design is provide us major benefit of using object oriented technology.
- Reuse reduces the cost and increases the value of both i.e. reusable components and the system.

Principle# 7: Think then Act

- Placing clear idea and complete thought before action will almost always produce better results.
- Thinking again the knowledge about how to do things right.
- If we follow these Hook's seven principles, many of the difficulties in building complex

Summary

- Software is defined as "computer programs, procedures, rules and possibly associated documentation and data pertaining to the operation of a computer based systems".
- The term software engineering is the systematic application of scientific and technological knowledge, methods, and experience to the design, implementation, testing, and documentation of software.
- Software Engineering (SE) is concerned with developing and maintaining software systems that behave reliably and efficiently, are affordable to develop and maintain, and satisfy all the requirements that customers have defined for them.
- The main characteristics of software are software is developed or engineered, software does not wear out and most software is custom-built, rather than being assembled from existing components.
- Today, software can be applied in various fields like, education, business, social sectors, etc. Software is designed to suite some specific goal or task like data processing, communication, information sharing etc.
- System Software is a collection of programs written to service other programs. It mainly deals with computer hardware and user. Example includes compilers, editors, file management utilities, operating systems, device drivers etc.
- Application software is the software that is designed to satisfy a particular need of a particular environment. Examples: student record software, railway reservation software, income tax software, word processors etc.
- Embedded Software resides within a product or system and is used to implement and control features and functions for the end-user and for the system itself.
- Real-time software programs that monitors / analyses / control real-world events as they occur is called real-time.
- Product-line Software is designed to provide a specific capability for use by many different customers or users.
- Web-based softwares are acts as an interface between the user and the Internet
- Engineering and scientific software applications range from astronomy to volcanology, from automotive stress analysis to space shuttle orbital dynamics and from molecular biology to automated manufacturing. Examples include CAD/CAM package, SPSS, MATLAB, Engineering Pro, Circuit analyzers etc.
- Software engineering is important because of the impact of large, expensive software systems and the role of software in safety-critical applications.
- A process was defined as a collection of work activities, actions, and tasks that are performed when some work product is to be created.

- A process framework establishes the foundation for a complete software process by identifying a small number of framework activities or tasks that are applicable to all software projects. Process framework encompasses a set of umbrella activities that are applicable across the entire software process.
- A generic process framework for software engineering defines five framework activities-communication, planning, modeling, construction, and deployment. In addition, a set of umbrella activities-project tracking and control, risk management, quality assurance, configuration management, technical reviews, and others—are applied throughout the process.
- Software engineering practice is a collection of principles, concepts, methods and tools that a software engineer calls upon on a daily basis.
- Software engineering practice is a broad array of principles, concepts, methods, approaches and tools that you must consider as software is planned and developed.
- Software engineering is a layered technology. Software engineering encompasses a process, technical methods, and use of tools to develop software products.
- Software process can be defined as a collection of procedures to develop the software product according to certain goals or standards.
- Software engineering methods provide the technical knowledge for building software.
- Principles that guide practice establish a foundation from which software engineering is conducted.
- David Hocker has proposed seven core principles that focus on software engineering practice.

Check Your Understanding

1. What is a Software ?
   (a) Software is set of programs.
   (b) Software is documentation and configuration of data.
   (c) Software is set of programs, documentation & configuration of data
   (d) None of the above
2. Which of the following is not the characteristic of software?
   (a) Software does not wear out (b) Software is flexible
   (c) Software is not manufactured (d) Software is always correct
3. Layers in layered technology of software are .....
   (a) Process, Methods, Tools (b) Internal, External, Central
   (c) Inner, Outer, Middle (d) Procedure, Method, Task

4. Which is not McCall's software quality factor?
   (a) Product Revision (b) Product Transition
   (c) Product Operation (d) Product Generation
5. Which is characteristic of software process?
   (a) Understanding (b) Visibility
   (c) Reliability (d) All of the above
6. Which of the five Generic Software Engineering Framework Activities?
   (a) Communication, Planning, Modelling, Construction, Deployment.
   (b) Communication, Risk Management, Measurement, Production, Reviewing
   (c) Analysis, Designing, Programming, Debugging, Maintenance
   (d) Analysis, Planning, Designing, Programming, Testing
7. Expert system, Robotics comes under _____.
   (a) Application software
   (b) Artificial Intelligent
   (c) Embedded software
   (d) Net sourcing
8. Which of the following is not present in layers of software?
   (a) Quality (b) Tools
   (c) Cost (d) Method
9. Which of these software engineering activities are not a part of s processes?
   (a) Software dependence (b) Software development
   (c) Software validation (d) Software specification
10. _____ and _____ are two kinds of software product.
    (a) CAD, CAM (b) Firmware, Embedded
    (c) Generic, Customised (d) none of the mentioned
11. Which of the following is not feature of legacy software?
    (a) Extensibility (b) Complex code
    (c) Flexibility (d) Poor documentation

ANSWER KEY

1. (c)	2. (d)	3. (a)	4. (d)	5.
6. (a)	7. (b)	8. (c)	9. (a)	10.
11. (d)				

Practice Questions

Q.I: Answer the following Questions in short.

1. What is software?
2. Name the types of software?
3. What is system software?
4. Name the McCall's factors.
5. What is software engineering?
6. What is software product and process?
7. Comment the statement: "Most Software is Custom Built, rather than being assembled from existing components".
8. Write umbrella activities of Software Process Framework?

Q.II: Answer the following Questions.

1. What is software engineering? Explain in detail.
2. Explain characteristics of software in detail.
3. Why there is need of Software Engineering?
4. Explain McCall's quality factors with diagram.
5. What is Software Process? Explain.
6. Write a note on: Essence of Practice.
7. Explain general principles of software engineering.
8. Describe software application domains in detail.
9. With the help of diagram describe generic software model.
10. Explain different layers in layered technology of software.

Q.III: Define the following terms:

1. Software
2. Software Engineering
3. Process
4. Software Quality
5. Product
6. Software process model
7. AI software
8. Product transition
9. Embedded software
10. Process Layer

Previous Exams Questions

BBA (CA)

Summer 2018

1. Define Software Engineering.

Ans. Refer to Section 2.2. (2M)

2. Explain different McCall's quality factors.

Ans. Refer to Section 2.7. (2M)

Winter 2018

1. Justify 'software does not wear out'.

Ans. Refer to Section 2.3 (2M)

2. Discuss Software Qualities Factors (McCall's Quality Factors).

Ans. Refer to Section 2.7 (4M)

Summer 2019

1. Define software Engineering.

Ans. Refer to Section 2.2 (2M)

2. Explain various characteristics of software engineering.

Ans. Refer to Section 2.3 (4M)

3. Short note: McCall's quality factors.

Ans. Refer to Section 2.7 (4M)

BCA (Science)

Summer 2018

1. Define software.

Ans. Refer to section 2.2 (1M)

2. Write a short note on McCall's Quality factors.

Ans. Refer to section 2.7 (5M)

3. Explain any four umbrella activities.

Ans. Refer to section 2.8 (4M)

4. Explain the essence of software engineering Practice.

Ans. Refer to section 2.10.1 (3M)

Winter 2018

1. Which of the following is not a McCall's quality factor under "product operation". (1M)

   (a) Reliability (b) Usability

   (c) Flexibility (d) Efficiency

Ans. Refer to section 2.7

2. Enlist the names of layers of Software Engineering. (1M)

Ans. Refer to section 2.5

3. Explain McCall's quality factors of "Product operation" group. (5M)

Ans. Refer to section 2.7

Summer 2019

1. Maintainability, Flexibility and Testability are factors of..... (1M)

   (a) Product operation (b) Product revision

   (c) Product transition (d) None of above

Ans. Refer to section 2.7

2. Explain General Principles of Software Engineering. (5M)

Ans. Refer to section 2.10.2

3. Define software engineering. Discuss its layers. (4M)

Ans. Refer to section 2.5

3 Software Development Life Cycle (SDLC) and Methodologies

Objectives...

- To understand SDLC Concepts and its activities.
- To study about Process Model.
- To learn Concurrent, Evolutionary and Prescriptive Process Models

3.1 INTRODUCTION

- A software life cycle is a series of identifiable stages that a software undergoes during its life cycle. As software life cycle is also often referred software process.
- To develop any software, it is first necessary to understand following two thing
  1. What is the purpose for which the software is being developed and What objectives it has to fulfil?
  2. What are the functions that the software has to perform to fulfil its objecti
- A key component of any software development process is the life cycle m which the process is based. Without this model, the development of a s product would not be in a systematic and disciplined manner.
- A successful software model is one that has been adopted by the software i and has undergone many modifications and enhancements to improve its eff and applicability. An ideal life cycle model is generic, flexible, adaptable, and s

3.2 SDLC

BBA(CA): S-19, W-18

- A software life cycle model is a particular abstraction that represents a software life cycle. A software life cycle model is often called a Software Development Life Cycle (SDLC).
- Software Development Life Cycle (SDLC) was developed by the National Computing Centre in 1960 for developing large scale functional business systems.
- A SDLC can be defined as, "a process or series of steps or phases that provide a model for the development of high quality software".
- There are different phases within SDLC, and each phase has its various activities. It makes the development team able to design, create, and deliver a high-quality product.
- SDLC concentrates on feasibility analysis, cost benefit analysis, project management hardware and software selection and personnel consideration.

3.2.1 Activities of SDLC

BCA: S-18,'19;BBA(CA): S-19

- The process used to create a software product from scratch to its public release is called Software Development Process (Life cycle) model.
- The characterized software development process models consist of three generic phases i.e., Definition, Development and Maintenance as shown in Fig. 3.1.

Fig. 3.1: Development Process Steps

- Software Development Life Cycle (SDLC) is simply a series of sequentially, interrelated activities leading to the successful completion of set of programs.
- The period of time during which these activities take place are called phases. Software development life cycle consists of several phases and these phases need to be identified along with defining the entry and exit criteria for every phase. A phase can begin only when the corresponding phase entry criteria are satisfied.
- The SDLC is a set of steps which are used for building a system.
- Various SDLC activities are as shown in Fig. 3.2.

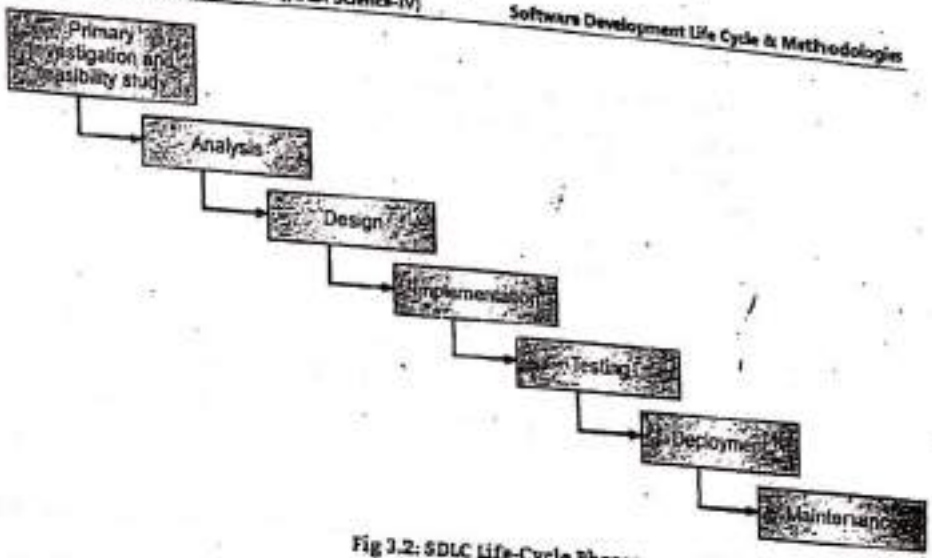


Fig 3.2: SDLC Life-Cycle Phases

- The steps/phases or activities involved in SDLC are explained below:

#1) Primary Investigation and Feasibility study

- Primary investigation which is also termed as preliminary system study is the first stage of system development life cycle.
- During this phase, all the relevant information is collected from the customer to develop a product as per their expectation. Any ambiguities must be resolved in this phase only.
- The feasibility study is basically the test of the proposed system through the point of its workability, meeting the user's requirements, effective use of resources and most importantly the cost effectiveness.
- During feasibility study, user determines the overall project scope, including economic, operational and human factors, identify key personnel, and develop timelines.

#2) Analysis or Requirement Gathering

- System Analysis is the process of studying a procedure or business in order to identify its goals and purposes and create systems and procedures that will achieve them in an efficient way.
- Business analyst and Project Manager set up a meeting with the customer to gather all the information like what the customer wants to build, who will be the end-user, what is the purpose of the product. Before building a product a core understanding or knowledge of the product is very important.

- For Example, A customer wants to have an application which involves money transactions. In this case, the requirement has to be clear like what kind of transactions will be done, how it will be done, in which currency it will be done, etc.
- Once the requirement gathering is done, an analysis is done to check the feasibility of the development of a product. In case of any ambiguity, a call is set up for further discussion.
- Once the requirement is clearly understood, the SRS (Software Requirement Specification) document is created. This document should be thoroughly understood by the developers and also should be reviewed by the customer for future reference.

#3) Design

- System design refers to the process of planning for a new system or for replacing the existing system.
- In simple words, systems analysis describes what the system should do whereas systems design focuses on how to achieve the objectives.
- In this phase, the requirement gathered in the SRS document is used as an input and software architecture that is used for implementing system development is derived.
- All the technical details such as programming languages and environments, machines, packages, application architecture, distributed architecture layering, memory size, platform, algorithms, data structures, global type definitions, interfaces, and many other engineering details are established in this design phase.
- The output of software design process is design documentation, pseudo codes, detailed logic diagrams, process diagrams, and detailed description of all functional or non-functional requirements of the software. The system design specifications serve as input for the next phase of the model.

#4) Implementation or Coding

- Implementation/Coding starts once the developer gets the Design document. The Software design is translated into source code. All the components of the software are implemented in this phase.

#5) Testing

- Testing starts once the coding is complete and the modules are released for testing. In this phase, the developed software is tested thoroughly and any defects found are assigned to developers to get them fixed.

- Retesting, regression testing is done until the point at which the software the customer's expectation. Testers refer SRS document to make sure software is as per the customer's standard.

#6) Deployment

- Once the product is tested, it is deployed in the production environmen UAT (User Acceptance testing) is done depending on the customer expecta
- In the case of UAT, a replica of the production environment is created customer along with the developers does the testing. If the customer f application as expected, then sign off is provided by the customer to go liv

#7) Maintenance

- After the software is deployed, then its maintenance begins.
- Once when the client starts using the developed systems, then the real issues and requirements to be solved from time to time.
- This procedure where the care is taken for the developed product is k maintenance.
- The typical software maintenance (in-contrast to any non-software produc not just fix the bugs or do changes but it may add new requirement which is up in the existing system.
- **System Evaluation:** Evaluation is the final check point of the SDLC; it is not feedback of system. There are three categories of this evaluation:
  - (i) **Developmental Evaluation:** This evaluation verifies the development and tools used. Also the system is developed on time and within budget.
  - (ii) Operational Evaluation. This evaluation mainly focused on Response system, user friendliness, Reliability of computation and adequacy of capacity.
  - (iii) **User Management Assessment Evaluation:** In this evaluation, che frequently management use the system and how far they are satisfied system.

3.2.2 Advantages of SDLC

- A formally defined method for software development in the form of t achieves a number of benefits:
  - A common vocabulary for each step.
  - Defined communication channels between development teams and stakeh
  - Clear roles and responsibilities among developers, designers, business a and project managers.
  - Clearly-defined inputs and outputs from one step to the next.

3.2.3 Difference between System Analysis and System Design

Sr. No	System Analysis	System Design
1.	System analysis is the process of examination of the problem.	System design is the creation as the system which is the solution of the problem.
2.	It is related with identifying all the constraints and influences.	It is related with the co-ordination of the activities, job procedures and equipment utilization in order to achieves system goals.
3.	It deals with data collection and a detailed evaluation of current system.	It deals with general and detailed design specification of input, output, files and procedures. It also deals with program development, testing and user acceptance.
4.	It gives logical model of the system through Data Flow Diagrams (DFDs) and Data Dictionary etc.	It provides technical specification and reports with which the problem can be tackled.

3.3 A GENERIC PROCESS MODEL (BCA: S-18, 19)

- A software process is a set of activities and associated results, which produce a software product. A software process model is an abstract representation of a software process.
- A process model is defined as, "a model of a process system that describes process organization, categorization, hierarchy, interrelationship, and tailorability".

OR

- IEEE defines, a process model as, "a framework containing the processes, activities and tasks involved in the development, operation and maintenance of a software product, spanning the life of the system from the definition of its requirements to the termination of its use."
- Software process models are systematic methods for controlling and co-ordinating the development of a software product achieving all the stated objectives.
- The generic software process models are abstractions of the process that can be used to explain different approaches to software development.
- This process model may be extended and adapted to create more specific software engineering processes.

3.3.1 Activities of Generic Process Model

- The generic process Model includes following five main activities.
  1. **Communication:** Involves communication among the customer and other stake holders; includes requirements gathering.
  2. **Planning:** Establishes a plan for software engineering work; addresses technical tasks, resources, work products, and work schedule.
  3. **Modelling (Analyze, Design):** Includes the creation of models to better under the requirements and the design.
  4. **Construction (Code, Test):** Combines code generation and testing to find errors.
  5. **Deployment:** Involves delivery of software to the customer for evaluation and feedback.

3.3.2 Advantages of Process Model

1. **Enables Effective Communication:** It enhances understanding and provides a specific basis for process execution.
2. **Facilitates Process Reuse:** Process development is a time consuming and expensive activity. Thus, the software development team utilizes the existing processes for different projects.
3. **Effective:** Since, process models can be used again and again; re-usable processes provide an effective means for implementing processes for software development.
4. **Facilitates Process Management:** They provide a framework for defining process status criteria and measures for software development.

3.3.3 Process Flow

- Process flow shows how the framework activities and the actions and tasks that occur within each activity are organized with respect to sequence and time.

Following are types of process flows in SDLC:

1. **Linear Process Flow**

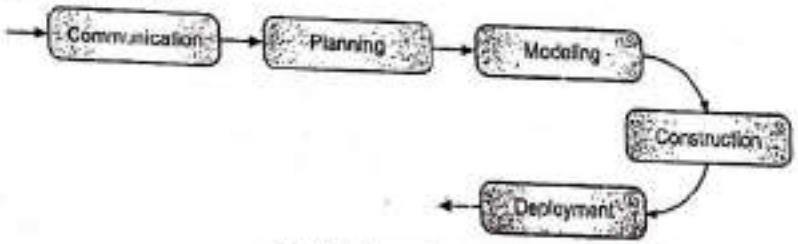


Fig. 3.3: Linear Process Flow

- A linear process flow executes each of the five framework activities in sequence, beginning with communication and culminating with deployment as shown in Fig. 3.3.

2. **Iterative Process Flow**

- An iterative process flow repeats one or more of the activities before proceeding to the next as shown in Fig. 3.4.

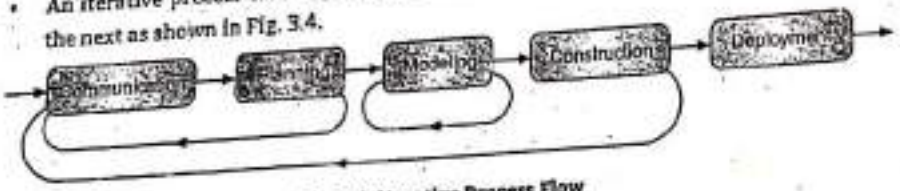


Fig. 3.4: Iterative Process Flow

3. **Evolutionary Process Flow**

- An evolutionary process flow executes the activities in a "circular" manner as shown in Fig. 3.5

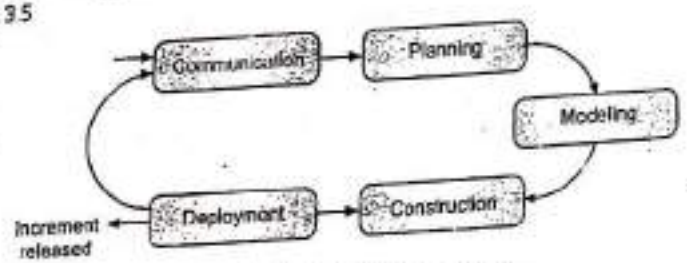


Fig. 3.5: Evolutionary Process Flow

4. **Parallel Process Flow**

- A parallel process flow executes one or more activities in parallel with other activities as shown in Fig. 3.6.

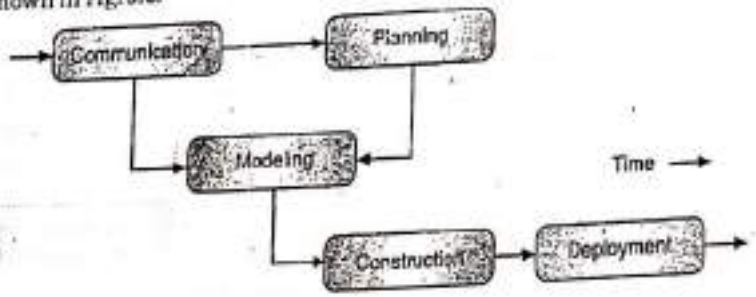


Fig. 3.6: Parallel Process Flow

3.3.4 Types of SDLC Process Model

- A Software Life Cycle Model is either a descriptive or prescriptive characterizatio how software is or should be developed.

1. **Descriptive model:** A descriptive model describes the history of how a partic software system was developed. A descriptive process model is defined as, "a m that describes 'what to do' according to a certain software process system".
- Descriptive models may be used as the basis for understanding and impro software development process or for building empirically grounded prescri models.
2. **Prescriptive model:** A prescriptive model prescribes how a new software sy should be developed.
- Prescriptive models are used as guidelines or frameworks to organize and stru how software development activities should be performed, and in which order.

3.4 PRESCRIPTIVE PROCESS MODEL

- A software process model presents a description of a process from some part perspective.
- A prescriptive process model defines a distinct set of activities, actions, milestones, and work products that are required to engineer high-quality softw
- The activities may be linear, incremental, or evolutionary. The process elemen the work products, framework, activities, quality assurance, action, task and c mechanism.
- There are three types of prescriptive process models. They are:
  1. Waterfall model.
  2. Incremental process model.
  3. Evolutionary process model.

3.4.1 Waterfall Model

(BCA: W-18, S-18; BBA(CA):

- The waterfall model was first process model. It is also referred to as a sequential life cycle model or 'Classic life cycle model'.
- This model is very simple to understand and use. It is traditional software li model and best understood by upper management.
- In waterfall model, a work flow is in a linear (sequential) fashion. The waterfall often used with well-defined adaptations or enhancements to current software

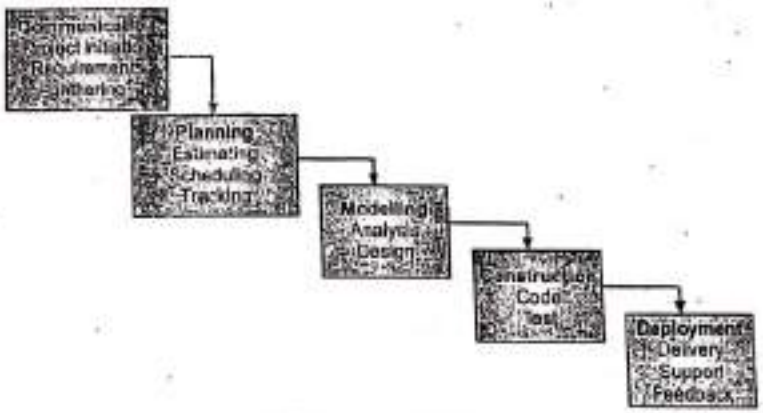


Fig. 3.7: Waterfall Model

[**NOTE:** The description of the phases of the waterfall model is same as that of the process model.]

- The waterfall life cycle model is defined as, "the model of building in stages, whereby one stage is completed before the next stage beings."
- In other words, in a waterfall model, each phase must be completed fully before the next phase can begin.
- This type of software development model is basically used for the project which is small and there are no uncertain requirements.
- In this model software testing starts only after the development is complete.
- In waterfall model phases do not overlap. At the end of each phase, a review takes place to check that the project is on the right path and whether or not to continue or discard the project.

Advantages:

1. Waterfall model is a linear model and of course, linear models are the most simple to be implemented.
2. The amount of resources required to implement this model is very minimal.
3. One great advantage of the waterfall model is that documentation is produced at every stage of the waterfall model development. This makes the understanding of the product designing procedure simpler.
4. After every major stage of software coding, testing is done to check the correct running of the code.
5. In waterfall model progress of system is measurable.

Disadvantages:

1. In waterfall model, there is the difficulty of accepting change after the process is in progress. Does not support iteration, so changes can cause confusion.
2. It has high amount of risks and uncertainty.
3. Requires customer patience because a working version of the program does not occur until the final phase.
4. It is a poor model for long and ongoing projects.

3.4.1.1 V and V Model

- A contemporary of traditional software development model is 'V-Model'. V-Model also referred to as the Verification and Validation Model. This is an extension of the Waterfall model.
- In this, each phase of SDLC must complete before the next phase starts. It follows a sequential design process same as the waterfall model. Testing of the device is planned in parallel with a corresponding stage of development.
- The crux of V model establishes an association between each phase of testing with that of development. The phases of testing are categorised as "Validation Phase" and that of development as "Verification Phase". Therefore for each phase of development there's a corresponding test activity planned in advance.

Architecture of V Model:

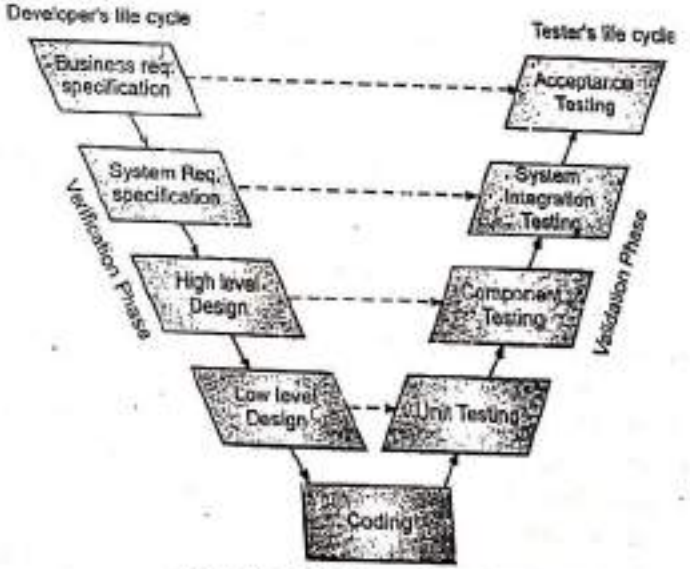


Fig. 3.8: Architecture of V and V Model

Phases of V Model:

Phases of Verification Phase of V-model:

1. **Business requirement analysis:** This is the first step where product requirements understood from the customer's side. This phase contains detailed communication to understand customer's expectations and exact requirements.
2. **System Design:** In this stage system engineers analyze and interpret the business of the proposed system by studying the user requirements document. System design is aimed at writing a detailed hardware and software specification.
3. **Architectural Design:** Architectural design is concerned with drafting the technical methodologies to be adopted with regard to completion of software development objectives. Architectural design is often termed as 'high-level design' which is aimed at providing an overview of solution, platform, system, product and service.
4. **Module Design:** In the module design phase, the system breaks down into small modules. The detailed design of the modules is specified, which is known as Low-Level Design.
5. **Coding Phase:** After designing, the coding phase is started. Based on the requirements, a suitable programming language is decided. There are some guidelines and standards for coding. Before checking in the repository, the final build is optimized for better performance, and the code goes through many code reviews to check the performance.

Phases of Validation Phase of V-model:

1. **Unit Testing:** In the V-Model, Unit Test Plans (UTPs) are developed during the module design phase. These UTPs are executed to eliminate errors at code level or unit level. A unit is the smallest entity which can independently exist, e.g., a program module. Unit testing verifies that the smallest entity can function correctly when isolated from the rest of the codes/ units.
2. **Integration Testing:** Integration Test Plans are developed during the Architectural Design Phase. The term 'integration testing' refers to collaborate pieces of code together to verify that they perform as a single entity.
3. **System Testing:** System Tests Plans are developed during System Design Phase. Unlike Unit and Integration Test Plans, System Tests Plans are composed by the client's business team.

   System Testing is performed when the complete system is ready, the application is then run on the target environment in which it must operate, and a conclusion

is drawn to figure out whether the system is capable of performing effic with least response time.

4. **Acceptance Testing:** Acceptance testing is related to the business requir analysis part. It includes testing the software product in user atmos Acceptance tests expose the compatibility problems with the different sy which is available within the user atmosphere. It discovers the non-fun problems like load and performance defects within the real user atmospher

When to use V-Model?

- When the requirement is well defined and not ambiguous.
- The V-shaped model should be used for small to medium-sized projects requirements are clearly defined and fixed.
- The V-shaped model should be chosen when sample technical resour available with essential technical expertise.

Advantages:

1. Easy to Understand.
2. Works well for small plans where requirements are easily understood.
3. Testing Methods like planning, test designing happens well before coding.
4. This model avoids the downward flow of the defects.
5. This saves a lot of time. Hence a higher chance of success over the w model.

Disadvantages:

1. Very rigid and least flexible.
2. Not a good for a complex project.
3. If any changes happen in the midway, then the test documents along with required documents, has to be updated.
4. Software is developed during the implementation stage, so no early proto the software are produced.

3.4.2 Incremental Process Model

- The incremental model is defined as, "a model of software development w product is designed, implemented and tested incrementally (a little more each time) until the product is finished."
- In incremental model, Multiple development cycles take place that mal software life cycle a multi-waterfall. In this model, cycles are divided up into more easily managed modules.

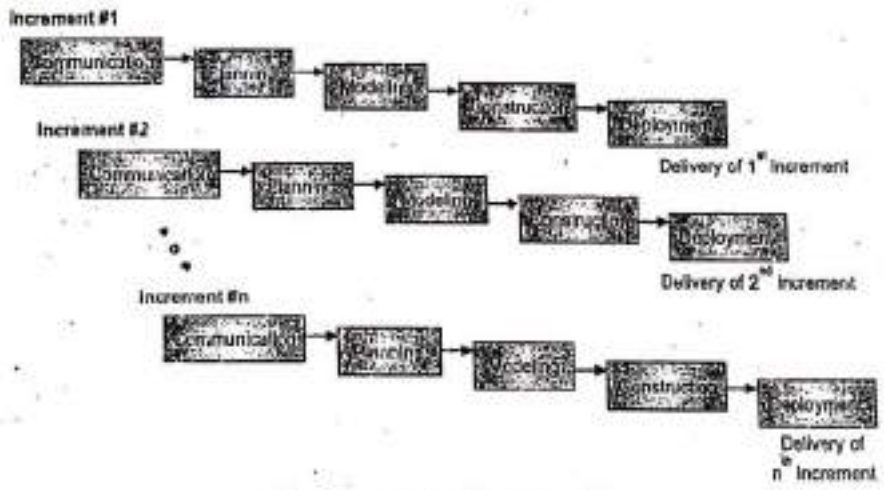


Fig. 3.9: Incremental Process Model

- The work flow is in a linear (sequential) fashion within an increment and is staggered between increments. Iterative nature of this model focuses on an operational product with each increment.
- The incremental model provides a needed set of functionality sooner while delivering optional components later.
- Incremental model used when requirements are well understood. In incremental model, multiple independent deliveries are identified.
- Incremental model is useful also when staffing is too short for a full-scale development.

Advantages:

1. Generates working software quickly and early during the software life cycle.
2. This model is more flexible to change scope and requirements.
3. It is easier to test and debug during a smaller iteration.
4. In this model, customer can respond to each built.
5. Lowers initial delivery cost.
6. Easier to manage risk because risky pieces are identified and handled during iteration.

Disadvantages:

1. Needs good planning and design.
2. Needs a clear and complete definition of the whole system before it can be broken down and built incrementally.
3. Total cost is higher than waterfall model.

3.4.3 Evolutionary Process Model

- All complex software system evolves over a period of time. Business and product requirement often changes as development proceeds.
- Evolutionary model are iterative. They are characterized in a manner that enables you to develop increasingly more complete version of the software.
- Evolutionary model is useful for projects using new technology that is not well understood. This is also used for complex projects where all functionality must be delivered at one time, but the requirements are unstable or not well understood at the beginning.
- It has basic two models i.e. Prototyping and Spiral Model.

3.4.3.1 Prototyping Model

(BCA: W-18, S-18, 19); (BBA(CA): S-19)

- Prototyping follows an evolutionary and iterative approach. It provides a working model to the user early in the process, enabling early assessment and increasing user's confidence. Prototyping is used when requirements are not well understood.
- The software prototyping refers to building software application prototypes which display the functionality of the product under development but may not actually hold the exact logic of the original software.
- Prototype (an early approximation of a final system or product and developed based on the currently known requirements) is a working model of software with some limited functionality. A prototype gives the user an actual view and feel of the system.
- The prototyping model serves to clarify requirements, which are not clear, hence reducing ambiguity and improving communication between the developers and users.
- It focuses on those aspects of the software that are visible to the customer/user. User feedback is used to refine the prototype.
- There is a great involvement of users in software development which helps in reducing risks associated with the software.

Phases of Prototyping Model:

- The following are the primary phases involved in the development cycle of any prototype model:
- **Requirement gathering:** In this approach product requirements are gathered. Developer and customer meet and discuss overall objectives of the software. Identify those requirements which are known. Detailed design like performance and security are not discussed in this step.

- **Quick Design:** Quick design system is created when requirements are known. The quick design focuses on the representation of those aspects of the software that are visible to the customer/user (e.g., input approaches and output formats.) rather then detailed design plan. It helps to construct a prototype.
- **Building Prototype:** Information gathered in quick design leads to the construction of a prototype. A prototype is constructed using several tools. Prototypes are prepared to represent input screen and output formats.
- **User Evaluation:** Evaluation of prototype recognizes its strengths and weaknesses like what is to be added or removed. User gives comments and suggestions which in turn are provided to the developer.
- **Refining prototype:** If the user is not satisfied with the current prototype then it improves according to the requirement and a new prototype is developed. New prototype is evaluated like previous prototype and the process continues until all requirements are met.
- When the user is satisfied with the developed prototype, a system is developed on the basis of final prototype.
- **Engineer product:** In this approach user accept the end prototype when all the requirements are completely met. It evaluates the end system with its regular maintenance for preventing its failures.

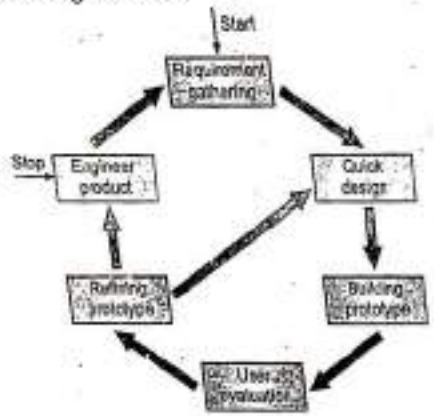


Fig. 3.10: Prototyping Process Model

Types of Prototyping Model:

- Four types of Prototyping model are:

1. **Rapid Throwaway Prototyping:** In this method, the prototype is developed rapidly based on the initial requirements and given to the client for review. Once

the client provides feedback, final requirements are updated and we final product begins. As the name suggests, the developed prototype is and it will not be part of the final product. It is also known as cl prototyping.

2. **Evolutionary Prototyping:** In this method, a prototype is made, and feedback is received. Based on the feedback, the prototype is refined client considers it the final product. It follows an incremental de approach and saves time compared to the rapid throwaway prototypin It is also known as breadboard prototyping.
3. **Incremental Prototyping:** In this type of prototype model, final requirements are break into smaller parts and each part is developed as prototype. In the end, all the parts (prototypes) are merged which becom product.
4. **Extreme Prototyping:** This type of prototyping model is mainly use applications. It is divided into three phases:
   (i) First basic prototype with static pages is created, it consists of HTML
   (ii) Using a prototype service layer, data processing is simulated.
   (iii) In the last phase, services are implemented.

Advantages:

1. Prototype model need not know the detailed input, output, processes, ad of operating system and full machine interaction.
2. In development procers of this model users are actively involved.
3. The development process is the best platform to understand the syste user.
4. Errors are detected much earlier.
5. Gives quick user feedback for better solutions.
6. In this model, missing functionality can be easily identified.
7. It also identifies confusing or difficult functions.

Disadvantages:

1. The client involvement is more and it is not always considered by the deve
2. It is a slow process because it takes more time for development.
3. Many changes can disturb the rhythm of the development team.

3.4.3.2 Spiral Model

(BCA: S-18; W-18; BBA (CA

- IEEE defines, the spiral model as, "a model of the software development pr which the constituent activities, typical requirements analysis, prelimin detailed design, coding, integration and testing are performed iteratively u software is complete".

- Spiral model was invented by Dr. Barry Boehm in 1988. It follows an evolutionary approach.
- Spiral Model is a combination of a waterfall model and iterative model. Each phase in spiral model begins with a design goal and ends with the client reviewing the progress.
- Inner spirals focus on identifying software requirements and project risks; may also incorporate prototyping.
- Outer spirals take on a classical waterfall approach after requirements have been defined, but permit iterative growth of the software.
- Risk analysis is the most important criteria considered in this model.
- The spiral model has five phases Communication, Planning, Risk Analysis, Modeling, Construction and Deployment as shown in Fig. 3.11.
- A software project repeatedly passes through these phases in iterations (called spirals in this model).
- The baseline spiral, starting in Communication phase, the Planning phase, requirements are gathered and risk is assessed, Modeling, Construction and Deployment. Each subsequent spiral builds on the baseline spiral.

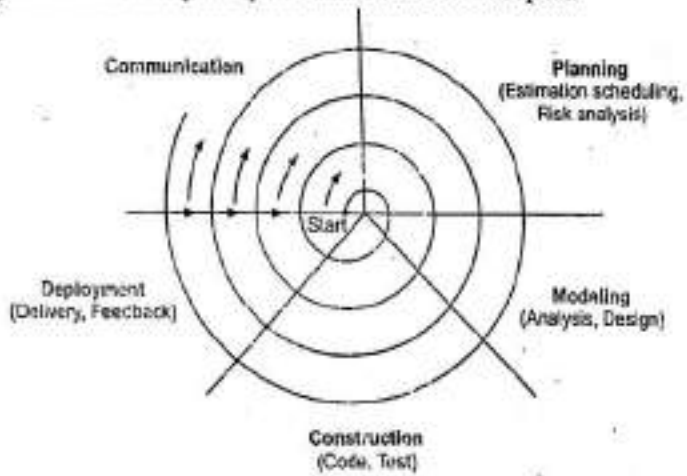


Fig. 3.11: Spiral Model

Advantages:

1. The spiral model accommodates life-cycle evaluation, growth and requirement changes.
2. It focuses on early error detection and design flaws.
3. It incorporates prototyping as a risk-reduction strategy.
4. The spiral model incorporates software quality objectives into the product.

Disadvantages:

1. Spiral model can be a costly model to use.
2. It is not suitable for low risk projects.
3. Risk analysis in this model requires highly specific expertise.

3.4.4 Comparative Analysis of Process Models

(BCA: S-19)

Table 3.1: Difference between various Process Models

Waterfall model	Simple and universal in nature.	Initial, once or twice.	Zero	Proven and universally understood.	Common knowledge	Solution
Prototyping model	Simple, but needs testing and confirmation on critical of aspects of technology solution.	Couple of times till prototype is approved.	Very low	Proven but needs testing	Technology	Prototype and solution
Incremental model	Calls progressive introduction of functions, features and implementation.	Very frequent; continuous basis for stage-wise confirmation	Medium	Configuration management.	High	Solution implementation strategy
Spiral model	Complex, large customer specifies, domain influences key, processes and technology to be tested.	Continues to move together.	High	Integration, handling of multiple technologies and solutions	Very high	Tools, technology solution implementation, customer participation.

CONCURRENT MODELS

- The concurrent process model shows the current state of activities, tasks and their associated states that remain in different phases.
- Fig. 3.12 shows a schematic representation of one Software engineering task within the modeling activity for the concurrent process model.
- The Analysis activity may be in any one of the states noted at any given time.
- All software engineering activities exis. concurrently but reside in different states, (For example, early in a project the communication activity has completed its first iteration and exists in the *Awaiting changes state*).
- The analysis activity in this model (which existed in the inactive state while initial communication was completed), now go to the *Under development state*.
- If, however, the customer indicates that changes in requirements must be made, then the modeling activity moves from the *Under development state* into the *Awaiting changes state*.

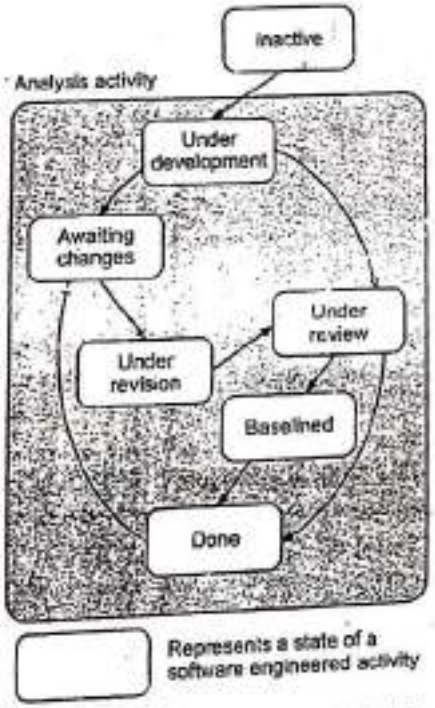


Fig. 3.12: Element of Concurrent Model

- The concurrent process model defines a series of events that will trigger transi from state to state for each of the Software engineering activities, actions, or tas
- This modeling is applicable for all types of software development and provid accurate picture of current state of software model.

Advantages:

1. It is flexible because of the number incremental releases can be determined project team.
2. This model gives immediate feedback from testing.
3. Each activity action or events in this model are exist simultaneously with activities.
4. This model has applicable for all kinds of development process.

Disadvantages:

1. Analysis and designs are complex task according to requirement. It requi remember the status of different activities.
2. They focus mainly on flexibility (and extensibility) and not on quality.

Summary

- A software development methodology or system development methodology framework that is used to structure, plan, and control the process of developi information system. Software methodologies are concerned with the proce creating software.
- A process model is a strategy (also known as software engineering para comprising process, methods, and tools layers as well as the general phase developing the software.
- Process flow describes how the framework activities and the actions and task occur within each framework activity are organized with respect to sequenc time.
- Various process flows are linear process flow (executes each of the five frame activities in sequence), iterative process flow (repeats one or more of the acti before proceeding to the next, evolutionary process flow (executes the activitie "circular" manner) and parallel process flow (executes one or more activiti parallel with other activities).
- Prescriptive process models were originally proposed to bring order to the cha software development. A descriptive model describes the history of how a parti software system was developed.

2. Explain Waterfall Model with a neat diagram. (5M)

Ans. Refer to section 3.4.1

3. Write any two advantages and two disadvantages of prototyping model. (4M)

Ans. Refer to section 3.4.3.1

4. Explain any two activities involved in "System Design" phase of SDLC model. (3M)

Ans. Refer to section 3.2

5. Differentiate between spiral model and prototype model. (4M)

Ans. Refer to section 3.4.3.3

Summer 2019

1. List the activities in SDLC. (2M)

Ans. Refer to section 3.2

2. Explain the generic process model. (5M)

Ans. Refer to section 3.3

3. Differentiate between system analysis and system design. (4M)

Ans. Refer to section 3.2

4. Differentiate between spiral and waterfall model. (4M)

Ans. Refer to section 3.4.3.3

◆◆◆

4... Requirement Engineering

Objectives...

- To understand meaning of Requirements and Requirement Engineering and its tasks.
- To know about Requirement gathering.
- To learn about feasibility study.
- To get information about various Fact Finding Techniques.

4.1 INTRODUCTION

- Designing and building computer software is challenging job. The software requirements are description of features and functionalities of the target system.
- **Requirement** is a condition possessed by the software or system component in order to solve real-world problems. The problems can be to automate a part of system, to correct the shortcomings of an existing system, to control a device and so on.
- IEEE defines requirement as, *"a condition or capability needed by a user to solve a problem or achieve an objective/goal."*

OR

- *"A condition or capability that must be met or possessed by a system or system component to satisfy a contract standard, specification or other formally imposed documents."*
- A requirement is a feature of the system or a description of something the system is capable of doing in order to fulfill the system's purpose.

Requirements Engineering:

- *Requirements engineering is a description of what the system will do without describing how it will do.*

- This is the process of gathering the software requirements from client, analyzing and document it.
- The aim of requirement engineering is to develop and maintain sophisticated and descriptive **'System Requirements Specification (SRS)'** document.

Requirement Gathering Process :

- In this process, the system analysts and engineers communicate with the client to know their ideas on what the software should provide, analyzing need and which features they want the software to include.
- Requirements received from client or end users are written in natural language.
- System analyst documents the requirement in technical language which can be used by the software development team.
- Requirement specifications are completed; the requirements mentioned in this document are then validated.

Requirement Engineering (RE) is the systematic use of proven principles, techniques, tools, standards and languages for the cost effective analysis, documentation and ongoing evolution of user needs and the specification of the external behavior of the system in order to satisfy these needs.

- Requirement engineering is defined as, "the systematic process of documenting requirements through an interactive co-operative process of analyzing the problem, documenting the resulting observations in a variety of representation format, and checking the accuracy of the understanding gain."

4.1.1 Types of Requirements

- The requirements are classified into three categories as shown in Fig. 4.1.

1. **Functional Requirements**

- IEEE defines function requirements as, "a function that a system or component must be able to perform".
- These requirements describe the interaction of software with its environment and specify the inputs, outputs, external interfaces and the functions that should be included in the software.

2. **Non-Functional Requirements**

- The non-functional requirements also known as quality requirements relate to system attributes, such as reliability and response time.
- Non-functional requirements arise due to user requirements, budget constraints, organizational policies etc.,

3. **Domain Requirements:**

- In software engineering, the requirements that are derived from the application domain of a system, instead from the needs of the users, are known as Domain Requirements.

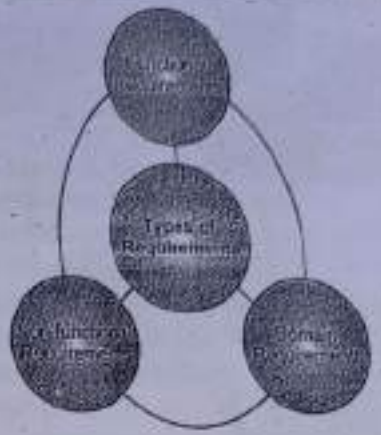


Fig. 4.1: Types of Requirements

- The domain requirements may be new functional requirements or specify a method to perform some particular computations.
- The software requirements are also classified as:

1. **User Requirements**
   - Written for customers
   - Collection of statements in a natural language, description of the services the system provides and its operational constraints.
2. **System Requirements**
   - Written for software developers that is a contract between client and contractor.
   - Structured document that gives the detailed description of the system services
   - It is Software Specification requirements the detailed software description that can serve as a basis for design or implementation. Typically it is written for software developers.

4.2 REQUIREMENT ENGINEERING TASKS (BCA: S-18, 19, W-18)

- **Requirement Engineering (RE) process:** The process to determine the requirement specification of the software is called the Requirement Engineering (RE) process as shown in Fig. 4.2.

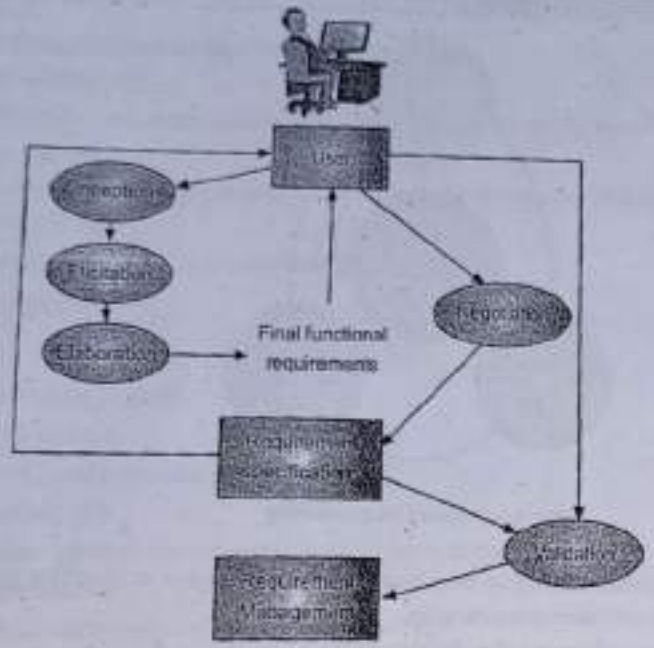


Fig. 4.2: Process of Requirement Engineering

- There are seven distinct tasks in requirement engineering. All these tasks or functions stress on the customer's needs must be satisfied. All these tasks collectively form the strong base for software design and construction.
- These tasks are: Inception, Elicitation, Elaboration, Negotiation, Specification, Validation, and Requirements Management.

4.2.1 Inception

- Inception means 'the commencement/beginning' and it tells about how does software project gets initialized. It establishes a basic understanding of the problem and the nature of the solution.
- During inception, the requirements engineer asks a set of following questions to know:
  1. Basic understanding of the problem.
  2. The people who want a solution.
  3. The nature of the solution that is desired.
  4. The effectiveness of preliminary communication and collaboration between the customer and the developer.

- Through these questions, the requirements engineer needs to:
  1. Identify the customers.
  2. Identified multiple viewpoints.
  3. Work toward collaboration.

4.2.2 Elicitation

- Elicitation means asking the customer, the user, and other people about the objectives for system or product. In other words,
- This is also known as the **gathering of requirements**. Here, requirements are identified with the help of customers and existing systems processes, if available.
- Eliciting requirements is difficult because of:
  1. **Problem of scope:** The customer gives the unnecessary technical detail rather than clarity of the overall system objective.
  2. **Problem of understanding:** Poor understanding between the customer and the developer regarding various aspect of the project like capability, limitation of the computing environment.
  3. **Problem of Volatility:** In this problem, the requirements change from time to time and it is difficult while developing the project.
- Elicitation may be accomplished through two activities that are Collaborative Requirements Gathering and Quality Function Deployment.

4.2.3 Elaboration

- Elaboration means refinement of requirement analysis and design.
- During elaboration, the software engineer takes the information obtained from customer during inception and elicitation is expanded and refined during elaboration.
- Elaboration focuses on developing a refined technical model of software functions, features, and constraints.
- It is an analysis modeling task. In this task:
  1. Use cases are developed.
  2. Domain classes are identified along with their attributes and relationships.
  3. State machine diagrams are used to capture the life on an object.
- The end result is an analysis model that defines the functional, informational, and behavioral domains of the problem.

4.2.4 Negotiation

- In this task, the software engineer reconciles the conflicts between what the customer wants and what can be achieved given limited business resources.

- Requirements are prioritized by the customers, users, and other stakeholders and assess the impact of each requirement on project cost and delivery time.
- Risks associated with each requirement are identified and analyzed.
- Using an iterative approach, requirements are eliminated, combined and/or modified so that each party achieves some measure of satisfaction.

4.2.5 Specification

- Specification is the final work product produced by the requirements engineer.
- It means different things to different people.
- It is normally in the form of a software requirements specification.
- It describes the function and performance of a computer-based system and the constraints that will govern its development.
- It formalizes the informational, functional, and behavioral requirements of the proposed software in both a graphical and textual format.
- A standard template should be developed and used.

4.2.6 Validation

- This task is concerned with ensuring that the gathered requirements in the software specification meet certain standards of quality.
- The specification is examined to ensure that:
  1. All software requirements have been stated definitely.
  2. That inconsistencies, omissions, and errors have been detected and corrected.
  3. The work products conform to the standards established for the process, the project, and the product.
- The formal technical review serves as the primary requirements validation mechanism include software engineers, customers, users, and other stakeholders.

4.2.7 Requirement Management

- Requirement management is defined as, "the process of controlling and tracking change in the requirements during the Requirements Engineering (RE) process and system development."
- During requirements management, the project team performs a set of activities to identify, control, and track requirements.
- There is change in requirements at any time as the project processed.
- Each requirement is assigned a unique identifier.
- The requirements are then placed into one or more traceability tables.
- These tables may be stored in a database that relates features, sources, dependencies, subsystems, and interfaces to the requirements.

4.3 ESTABLISHING GROUNDWORK FOR AN UNDERSTANDING OF SOFTWARE REQUIREMENT

- Initially requirement gathering for the project is done where various stakeholders and software engineers work together as the single team on the project. Similarly need to consider the following facts about clients :
  - May be located at different locations.
  - May have little idea of what is required.
  - May have different and conflicting opinions about the system to be built.
  - May be lacking technical knowledge.
  - May have insufficient time to interact with the requirement gathering engineer.
- **Requirement Engineering** is actually a matter of conducting meaningful conversations with colleagues who are members of the team.
- The conversations and meetings should happen by overcoming the physical distance barrier and the time barrier too.

Tasks as the Groundwork (basic work):

- Following are various tasks that are to be carried out as the groundwork before actually the requirement gathering work starts.
  1. **To identify the various Stakeholders in the Project:** The foremost and important task in the groundwork is to identify the various stakeholders in the project.
     - **Stakeholder** is anyone who benefits in a direct or indirect way from the system which is being developed.
     - Various sections managers like, Business operations managers, product managers, marketing people, internal and external customers, end users, consultants, product engineers, software engineers, testing team, support and maintenance engineers, and others all are considered as the stakeholders.
  2. **To recognize Multiple Viewpoints:** Next task is to recognize multiple viewpoint of the stakeholders about the system so that all the different perspectives abo any single requirement can be identified and studied and consolidated to sing requirement.
  3. **Working toward collaboration:** To achieve the targets in the specified time a infra-structure limit the work should be carried out in the collaborative man For this periodic meetings and feedback sessions are to be conducted in a plan manner.
  4. **Asking first questions to the stakeholders:** The first questions to stakeholders are really of great importance because they give the basic behind the work to be done and help to identify all the stakeholders of the sys

Ask the First Questions such as,

- Who is behind the request for this work ?
- Who will use the solution? What will be the economic benefit of a successful solution?
- Is there another source for the solution that one needs? etc.

4.4 REQUIREMENT GATHERING

- Software requirement gathering is subdomain of software requirement engineering of software engineering.

Requirement gathering / Elicitation:

- Involves customer interaction with the development team.
- Collects all relevant information regarding the software which is to be developed from the user requirement.
- Combines all the aspects of problem solving, elaboration, negotiation, and specification.
- All the stakeholders work together to identify the problem, propose elements of the solution, negotiate and compromise on different approaches and then specify a preliminary set of solution requirements.
- There are a number of requirements elicitation methods. Few of them are listed below:

1. Collaborative Requirements Gathering

- Collaborative requirements gathering have following basic guidelines:
  - (i) Rules for preparation and participation are established.
  - (ii) Meetings are conducted and attended by both software engineers and other stakeholders.
  - (iii) A "facilitator" (can be a customer, a developer etc.) controls the meeting.
  - (iv) An agenda is suggested that is formal enough to cover all important points but informal enough to encourage the free flow of ideas.
  - (v) A "definition mechanism" (can be work sheets, flip charts, chat room, etc.) is used.
- The goal is to identify the problem, propose elements of the solution, negotiate different approaches, and specify a preliminary set of solution requirements in an atmosphere i.e., conducive to the accomplishment of the goal.
- To better understand the flow of events as they occur, we present a brief scenario that outlines the sequence of events that lead up to the requirements gathering meeting, occur during the meeting, and follow the meeting.
- During inception basic questions and answers establish the scope of the problem and the overall perception of a solution.

2. Quality Function Deployment (QFD)

- QFD is a quality management technique that translates the needs of the customer into technical requirements for software.
- QFD concentrates on maximizing customer satisfaction from the software engineering process.
- To accomplish this, QFD emphasizes an understanding of what is valuable to the customer and then deploys these values throughout the engineering process.
- QFD identifies three types of requirements as explained below:
  - (i) **Normal Requirements** the objectives and goals that are stated for a product or system during meetings with the customer. If these requirements are present, the customer is satisfied. Examples of normal requirements might be requested types of graphical displays, specific system functions, and so on.
  - (ii) **Expected Requirements** are implicit to the product or system and may be so fundamental that the customer does not explicitly state them. Their absence will be a cause for significant dissatisfaction. Examples of expected requirements are ease of human/machine interaction, overall operational correctness and reliability and so on.
  - (iii) **Exciting Requirements** features go beyond the customer's expectations and prove to be very satisfying when present. For example, software for a new smart phone comes with standard features, but is coupled with a set of unexpected capabilities (like multi-touch screen) that delight every user of the product.
- Although QFD concepts can be applied across the entire software process specific QFD techniques are applicable to the requirements elicitation activity.
- QFD uses customer interviews and observation, surveys, and examination of historical data (e.g., problem reports) as raw data for the requirements gathering activity. These data are then translated into a table of requirements called the customer voice table i.e., reviewed with the customer and other stakeholders. A variety of diagrams, matrices, and evaluation methods are then used to extract expected requirements and to attempt to derive exciting requirements.

3. Usage Scenarios:

- As requirements in requirement gathering are gathered, an overall vision of system functions and features begins to materialize. However, it is difficult to move into more technical software engineering activities until you understand how these functions and features will be used by different classes of end users.
- To accomplish this, developers/engineer and users can create a set of scenarios that identify a thread of usage for the system to be constructed. The scenarios, often called use cases, provide a description of how the system will be used.

Elicitation Work Products:

- The work products produced as a consequence of requirements elicitation will vary depending on the size of the system or product to be built.
- For most systems, the work products include following:
  - (i) A bounded statement of scope for the system or product.
  - (ii) A statement of need and feasibility.
  - (iii) Any prototypes developed to better define requirements.
  - (iv) A set of usage scenarios that provide insight into the use of the system or product under different operating conditions.
  - (v) A list of customers, users, and other stakeholders who participated in requirements elicitation.
  - (vi) A description of the system's technical environment.
  - (vii) A list of requirements (preferably organized by function) and the domain constraints that applies to each.

4.5 FEASIBILITY STUDY (BCA: S-18, 19)

- Feasibility is the practical extent to which a project can be performed successfully.
- To evaluate feasibility, a feasibility study is performed, which determines whether the solution considered to accomplish the requirements is practical and workable in the software.
- Information such as resource availability, cost estimation for software development, benefits of the software to the organization after it is developed and cost to be incurred on its maintenance are considered during the feasibility study.
- The objective/goal of the feasibility study is to establish the reasons for developing the software that is acceptable to users, adaptable to change and conformable to established standards. Various other objectives/goals of feasibility study are listed below.

1. To analyze whether the software will meet organizational requirements.
2. To determine whether the software can be implemented using the current technology and within the specified budget and schedule.
3. To determine whether the software can be integrated with other existing software.

4.5.1 Types of Feasibility (BCA: W-18)

- Various types of feasibility that are commonly considered are: Technical Feasibility, Operational Feasibility, and Economic Feasibility as shown in Fig. 4.3.

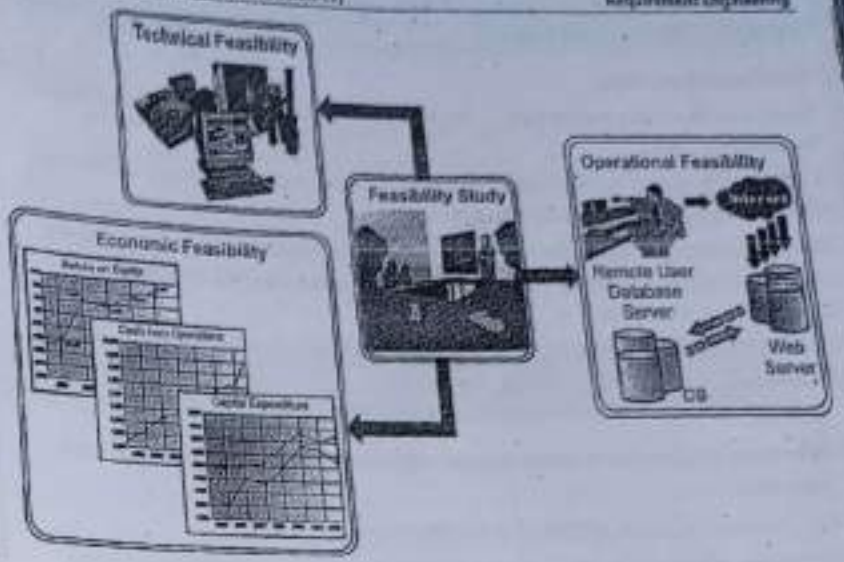


Fig. 4.3: Types of Feasibility

- These types of feasibility in Fig. 4.3 are explained below:

1. Technical Feasibility:

- Technical feasibility refers, "to the technical resources needed to develop, purchase, install, or operate the system".
- Technical feasibility assesses the current resources (such as hardware and software) and technology, which are required to accomplish user requirements in the software within the allocated time and budget.
- For this, the software development team determines whether the current resources and technology can be upgraded or added in the software to accomplish specified user requirements.
- Technical feasibility also performs the following tasks:
  - (i) Analyzes the technical skills and capabilities of the software development team members.
  - (ii) Determines whether the relevant technology is steady and established.
  - (iii) Ascertains that the technology chosen for software development has a large number of users so that they can be consulted when problems arise or improvements are required.

2. Operational Feasibility:

- Operational feasibility means that, "a proposed system will be used effectively after it has been developed".
- It assesses the extent to which the required software performs a series of steps to solve business problems and user requirements.
- This type feasibility is dependent on human resources (software development team) and involves visualizing whether the software will operate after it is developed and be operative once it is installed.
- Operational feasibility also performs the following tasks:
  - (i) Determines whether the problems anticipated in user requirements are of high priority.
  - (ii) Determines whether the solution suggested by the software development team is acceptable.
  - (iii) Analyzes whether users will adapt to new software.
  - (iv) Determines whether the organization is satisfied by the alternative solutions proposed by the software development team.

3. Economic Feasibility:

- It determines whether the required software is capable of generating financial gains for an organization.
- It involves the cost incurred on the software development team, estimated cost of hardware and software, cost of performing feasibility study, and so on.
- For this, it is essential to consider expenses made on purchases (such as hardware purchase) and activities required to carry out software development. In addition, it is necessary to consider the benefits that can be achieved by developing the software.
- Software is said to be economically feasible if it focuses on the issues listed below:
  - (i) Cost incurred on software development to produce long-term gains for an organization.
  - (ii) Cost required to conduct full software investigation (such as requirements elicitation and requirements analysis).
  - (iii) Cost of hardware, software, development team, and training.

4.6 FACT FINDING TECHNIQUES (BCA: S-19, W-18)

- To develop any system the user (systems analyst) needs to do collect facts and all relevant information. The facts when expressed in quantitative form are termed as data.

- The success of any project is depended upon the accuracy of available data. Accurate information can be collected with help of certain techniques. These specific techniques for finding information of the system are termed as Fact Finding Techniques.
- Fact finding is the formal process of using research, interviews, questionnaires, and other techniques to collect information about systems, requirements, and preferences.
- It is also called as Information Gathering or Data Collection.

4.6.1 Interviews (BCA: S-18, W-1

- Interview is the best method for producing qualitative information such as opinion polices, subjective descriptions of activities and problems.
- Interviews are strong medium to collect requirements from individuals or fr group.
- In this method, the analyst sits face to face with the people and records th responses.
- During the interview, the respondents and analyst discuss with each other.
- This method use to find the area of misunderstanding, unrealistic expectations.
- Using question-answer format, analyst collects the general information about system.
- The information collected is quite accurate and reliable as the interviewer can c and cross check the doubts there itself.

Types of Interviews:

- There are main two types of interviews i.e. Structured and Unstructured as expla below:

1. Structured (Closed) Interviews: In this type, every single information to gath decided in advance.

Advantages:

1. These interviews follow pattern and matter of discussion firmly.
2. Structured interview ensures uniform wording of questions for all responde
3. It is easy for analyst to evaluate.
4. It contains the set of prescribed answer, resultant into shorter interview.
5. For this interviewer required limited training.

Disadvantages:

1. Cost of preparation of uniform question is high.
2. High level of structure may not be suitable for all situations.

3. This interview reduces the respondent's spontaneity and ability of the interview to follow up the comments of interviewee.

2. **Unstructured (Open) Interview:** In this type the information to gather is not decided in advance, more flexible and less biased.

Advantage:

1. In Unstructured interviews, the respondents are free to answer in their own words. In this way their views are not restricted. So the interviewer gets a larger area to further explore the issues relating to a problem.

Disadvantages:

1. It may happen that the interview goes in some unwanted direction and the basic facts for which the interview was organized do not get relieved.
2. Analysis and interpretation of results may be lengthy.
3. It may take extra time to collect required facts.

- Apart from these there are some more interview techniques like Oral interviews, Written interviews etc.
- The advantage of interviews is that the analyst has a free hand and he can extract almost all the information from the respondents but it is a very time consuming method. He should also employ other means such as questionnaires, record reviews, etc.

4.6.2 Questionnaires

(BCA: S-19, BBA(CA): S-19)

- It is the technique used to extract information from large number of people.
- This technique can be adopted and used only by a skillful system analyst.
- A document with pre-defined set of objective questions and respective options is handed over to all customers to answer, which are collected and compiled.

Advantages:

1. The questionnaire contained the series of questions framed together in logical manner. These questions are simple, clear and to the point.
2. The questionnaire can send to people by email or by post.
3. This is the cheapest source of fact finding.

Disadvantage:

1. A drawback of this method is, if an option for some issue is not mentioned in the questionnaire, the issue might be left unattended.

- **Example:** We can get basic information of students using following questionnaire:

Name :

Address :

Qualification :

Contact Number :

Email : @

Gender : Male ☐ Female ☐

Hobbies :

Achievements :

4.6.3 Record View

- Records and reports are the collection of information and data about the system and its operations.
- The information related to the system is available in documents, newspapers, magazines, journals, electronic media etc.
- This record review helps the analyst to get valuable information about the system and the organization.
- The analyst may examine the records either at the beginning of the study which will give him a fair and perfect data about the system.

- It is easy to understand the system with actual operations.
- **Example:** Employee attendance record, Salary record, Service book can be observed by analyst to know how to manage the requirements in the system.

4.6.4 Observation

- Unlike the other fact finding techniques, in this method the team of experts visit the customer's organization or workplace.
- Analysts observe the actual working of the existing installed systems or manual system.
- They observe the workflow of document at customer's end and how execution problems are dealt.
- The experts may observe the unwanted things from existing system as well and simply cause delay in the development of the new system.
- The experts team itself finds some conclusions which help to form requirements expected from the software.

4.7 SOFTWARE REQUIREMENTS SPECIFICATION (SRS)

- A software requirements specification (SRS) is a document that captures complete description about how the system is expected to perform. The SRS needs to be correct, complete, consistent, updatable, verifiable etc. It is usually signed off at the end of requirements engineering phase.

Characteristics of SRS:

- Various characteristics of SRS are:
  1. **Correct:** The SRS should be made up to date when appropriate requirements are identified.
  2. **Unambiguous:** When the requirements are correctly understood then only it is possible to write an unambiguous SRS.
  3. **Complete:** To make the SRS complete, it should be specified what a software designer wants to create a software.
  4. **Consistent:** It should be consistent with reference to the functionalities identified.
  5. **Specific:** The requirements should be mentioned specifically.
  6. **Traceable:** What is the need for mentioned requirement? This should be correctly identified.

Types of Specification:

- The types of requirements which are captured during SRS are as follows:

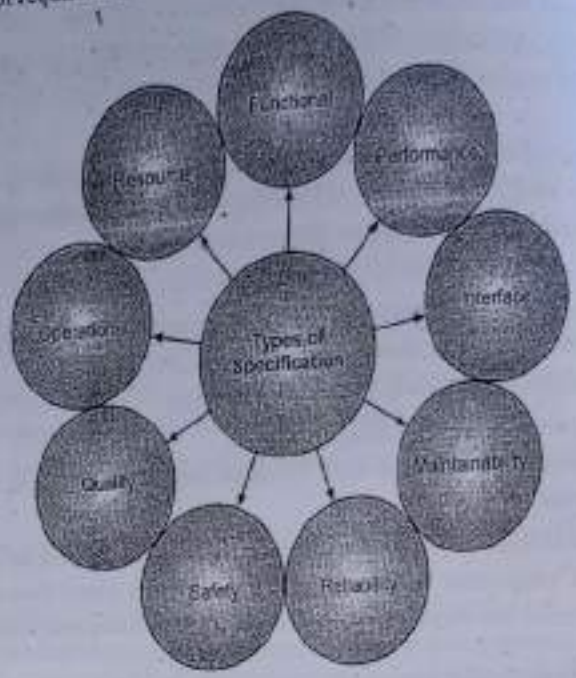


Fig. 4.4: Types of Specification

Summary

- Requirement is defined as a condition or capability needed by a user to s problem or achieve an objective.
- Requirements are the basis of all software applications as they focus on wh application must do. Requirements are elicited from the customer outlining wh wish the system to do, and recorded in a language the customer understands.
- The various types of requirements are User requirements, System require Functional requirements, and Non-functional requirements and so on.
- The individual who performs the system investigation and who may or may related to computer programming is called as System Analyst.
- The feasibility study activity involves the analysis of the problem and collectio relevant information relating to the systems products such as the differe items which would be input to the system, the processing required to be carr

on these data, the output data required to be produced by the system as well as various constraints on the behaviour of the system.

- Various types of feasibility are Technical feasibility, Economic feasibility and Operational feasibility.
- The specific techniques analyst used for collecting data about requirements are called as Fact Gathering Techniques.
- Interview, Questionnaire, Record view and Observations are the different fact finding techniques used by the analyst.
- RE (Requirement Engineering) provides an appropriate mechanism of understanding what the customer or user wants, assessing feasibility, analyzing need, negotiating a reasonable solution, managing the requirements, validating the specification and specifying the solution as they are transformed into an operational system.
- Inception also known as beginning.
- The meaning of elaboration is 'to work out in detail'.
- Negotiation is discussion on financial and other commercial issues. Negotiation function is not unusual for customer to ask for more than can be achieved, given limited business resources.
- Specification serves as a foundation of all subsequent of software engineering activities.
- An interview is most commonly used, and normally most useful, fact-finding technique. Interview enables collection of information from individuals face-to-face. Interview objectives include finding out facts, verifying facts, clarifying facts, generating enthusiasm, getting end-user involved, identifying requirements, and gathering ideas and opinions.
- Structured interviews are very formal. In a structured interview, each candidate is asked similar questions in a predetermined format.
- Unstructured interviews are much more casual and unrehearsed.
- The Questionnaire consists of series of questions framed together in logical manner. The questions are simple, clear and to the point.
- Many kinds of records and reports can provide valuable information about organization and operation. In Record Reviews, analyst examines information that has been recorded about the system and about the users. The information related to the system is published in the sources like newspapers, magazines, journals, documents etc.
- In Observations method, the analyst himself visits the organization. He observes, understand the flow of documents, working of the existing system, the users of the system etc.

Check Your Understanding

1. The process to determine the requirement specification of the software is called the _____ process.
   (a) Software Engineering (b) Requirement Engineering
   (c) System Engineering (d) System analysis
2. Which is not a task of Requirement Engineering?
   (a) Design (b) Elicitation
   (c) Inception (d) Elaboration
3. Which is the type of feasibility study?
   (a) Technical Feasibility (b) Operational Feasibility
   (c) Economic Feasibility (d) All of the Above
4. _____ is the technique used to extract information from large number of people.
   (a) Record view (b) Questionnaires
   (c) Observation (d) None of the above
5. _____ is the final work product produced by the requirements engineer.
   (a) Negotiation (b) Elicitation
   (c) Specification (d) Inception

ANSWER KEY

1. (b)	2. (b)	3. (a)	4. (c)	5. (c)

Practice Questions

Q.I: Answer the following Questions in short.

1. What is requirement? Enlist types of requirements.
2. What is inception?
3. Define questionnaire. Give its types.
4. What is elaboration?
5. Name the four fact finding techniques.
6. What is requirement management? Define it.
7. Which are types of feasibility?
8. Explain structured interview.

Q.II: Answer the following Questions.

1. Explain in detail the tasks of the requirement engineering.
2. What is the need of the requirement engineering?

3. Explain fact finding techniques in details.
4. Write a difference between structured and unstructured interview.
5. Explain problems arise as elicitation occurs?
6. What is requirement engineering? Explain with its advantages.
7. Describe requirement gathering in detail.
8. Which are the characteristics of SRS?

Q.III: Define the following terms.

1. Interview
2. Questionnaires
3. Record Review
4. Observation
5. Requirement
6. Economic feasibility
7. Functional Requirement
8. Requirement Management
9. Specification task
10. Negotiation task

Previous Exams Questions

BBA(CA)

Summer 2018

1. Define Economical feasibility. [2M]

Ans. Refer to Section 4.5.

2. What is fact finding technique ? [2M]

Ans. Refer to Section 4.6.

Winter 2018

1. What are the different types of interviewing ? [2M]

Ans. Refer to Section 4.7.1

2. Explain fact finding technique in detail. [4M]

Ans. Refer to Section 4.7

3. Write short note : Requirement Anticipation [4M]

Ans. Refer to Section 4.3

Summer 2019

1. Define questionnaire. Give its types [2M]

Ans. Refer to Section 4.7.2

2. Discuss different fact finding techniques. [4M]

Ans. Refer to Section 4.6

BCA (Science)

Summer 2018

1. _____is the primary requirements validation mechanism.
   (a) Negotiation (b) Formal Technical Review
   (c) Elaboration (d) Specification

Ans. Refer to section 4.2

2. Which of the following is not a fact finding technique?
   (a) Specification (b) Interview
   (c) Questionnaire (d) Observation

Ans. Refer to section 4.6

3. List and Explain any 4 tasks involved in Requirement Engineering.

Ans. Refer to section 4.2

4. Compare structured interview with unstructured interview.

Ans. Refer to section 4.6.1

5. Write a short note on feasibility study.

Ans. Refer to section 4.5

Winter 2018

1. SRS stands for_____.
   (a) Software Request Specification
   (b) Software Requirement Specification
   (c) System request specification
   (d) System requirement specification

Ans. Refer to section 4.2

2. Explain any two types of feasibility study.

Ans. Refer to section 4.5.1

3. Explain any two tasks of requirement engineering.

Ans. Refer to section 4.2

4. Differentiate between structured and unstructured interview. [3M]

Ans. Refer to section 4.6.1

5. With the help of diagram describe Requirement Engineering process. (RE Process) [4M]

Ans. Refer to section 4.1

Summer 2019

1. ..... means asking the customer, the user and other people about objectives for system/product. [1M]
   (a) Elaboration (b) Elicitation
   (c) Inception (d) Negotiation

Ans. Refer to section 4.2

2. Define specification. [1M]

Ans. Refer to section 4.2.5

3. What is feasibility study? Explain any one type in detail. [4M]

Ans. Refer to section 4.5

4. Write a short note on questionnaire. [3M]

Ans. Refer to section 4.5.2

5. Explain the prototyping model with its advantages and disadvantages. [5M]

Ans. Refer to section 4.3.1

❖❖❖

5 Analysis and Design Tools

Objectives...

- To understand basic concepts of Analysis and Design Tools
- To learn concepts of Decision Trees and Decision Tables
- To study Data Flow Diagrams (DFDs) and Data Dictionary
- To understand concepts of Input Design, Output Design and Pseudocode
- To know the meaning of cohesion and coupling in structured design

5.1 INTRODUCTION

- Software analysis and design include all activities, which help the transformation of requirement specification into implementation.
- Requirement specifications specify all functional and non-functional expectations from the software.
- These requirement specifications come in the shape of human readable and understandable documents, to which a computer has nothing to do.
- Software analysis and design is the intermediate stage, which helps human-readable requirements to be transformed into actual code.
- Analysis and design tools enable a software engineer to create models of the system to be built.
- The models contain a representation of data function, and behavior (at the analysis level) and characterizations of data, architectural, component level, and interface design.
- By performing consistency and validity checking on the models, analysis and desig tools provide a software engineer with some degree of insight into the analys

representation and help to eliminate errors before they propagate into the design, or worse, into implementation itself.

5.2 DECISION TREE AND DECISION TABLE

- Decision-making is an integral part of any organization no matter how small, simple or big and complex it may be. Thus, decisions have to be made and set procedures are to be followed as the subsequent actions.
- A process specification describes the purpose of processes depicted in the DFD, the inputs to the process, and the output. It describes what the process should do and not how it should do it.
- However, if the process relates to some decision making then the decision rules used in the process may be described. The decision rules are better represented by documentation tools such as Decision Table and Decision Tree.
- A decision table (also known as a logic table) and a decision tree (also known as a decision diagram) describe the conditions associated with particular actions or decisions, along with the constraints on the associated behavior.

5.2.1 Decision Tree

- A decision tree is a decision support analysis and design tool that uses a tree-like graph or model of decisions and their possible consequences.
- Decision tree is a diagram that represents conditions and actions sequentially. It shows which condition to consider first, which second and so on. It also shows the relationship of each condition and its permissible actions.
- In other words, in a decision tree each branch node represents a choice between a number of alternatives, and each leaf node represents a classification or decision.
- The decision sequence starts from the root of the tree that is usually on the left of the diagram, (See Fig. 5.1). The path to be followed to traverse the branches is decided by the priority of the conditions and the respectable actions. A series of decisions are taken, as the branches are traversed from left to right.
- The nodes are the decision junctions. After each decision point there are next set of decisions to be considered. Therefore at every node of the tree represented conditions are considered to determine which condition succeeds before moving further on the path.
- A decision tree can be defined as, "a graphical representation of specific decision situations in a structured decision process".

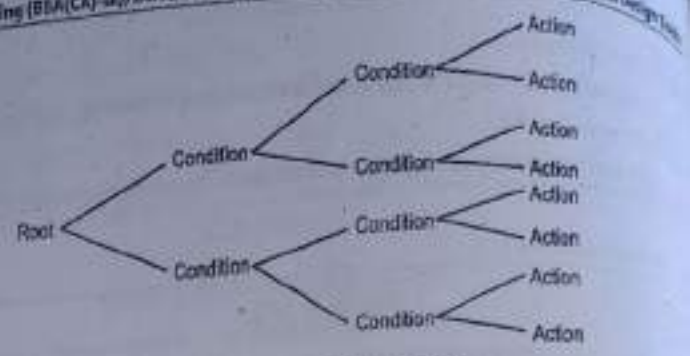


Fig. 5.1: Basic Format of Decision Tree

Examples for Decision Trees:

- **Example 1:** Consider the Discount Policy of a saree manufacturer for his custom According to the policy the saree manufacturer give discount to his customers ba on the type of customer and size of their order. For the individual, only if the or size is 12 or more, the manufacturer gives a discount of 50% and for less tha sarees the discount is 30%.

Whereas in case of shopkeepers or retailers, the discount policy is different. If order is less than 12 then there is 15% discount. For 13 to 48 sarees order, the disc is 30%, for 49 to 84 sarees 40% and for more than 85 sarees the discount is 50%.

The decision policy for discount percentage can be put in the form of a decision displayed in Fig. 5.2.

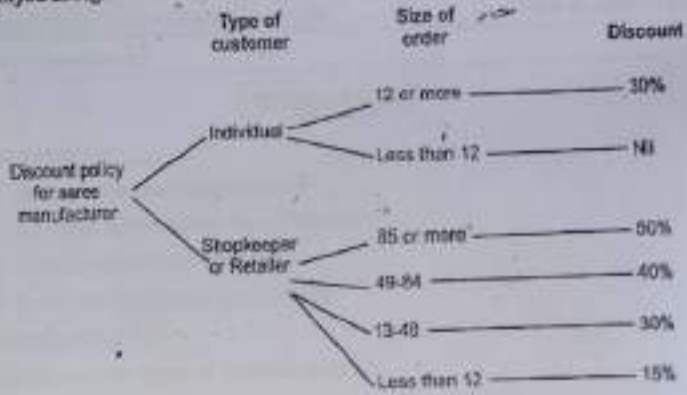


Fig. 5.2: Decision tree for Example 1

- **Example 2:** Bookstores get a trade discount of 25%; for orders from libraries individuals, 5% allowed on orders of 6-19 copies per book title; 10% on orders fo

... copies per book title; 15% on orders for 50 copies or more per book title. The decision tree is as shown in Fig. 5.3.

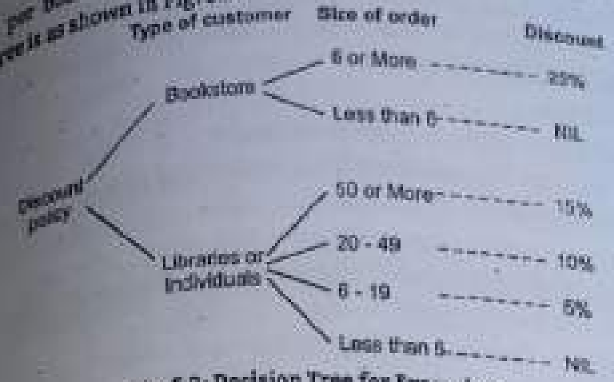


Fig. 5.3: Decision Tree for Example 2

Example 3: Consider the company which gives discounts based on amounts on three different values as follows:

- If payment is made within 10 days and the purchasing amount more than 10,000, the company offers 3% discount on the invoice. If the purchasing amount is between 5000 to 10,000 then the company offers 2% discount and if amount < 5000 then no discount offer and customer has to pay full invoice amount. If customer is not paying within 10 days then if he is regular customer then 1% discount is offer for amount more than 10,000 else no discount is offer and customers have to pay full invoice amount.

Draw a decision tree for the above problem.

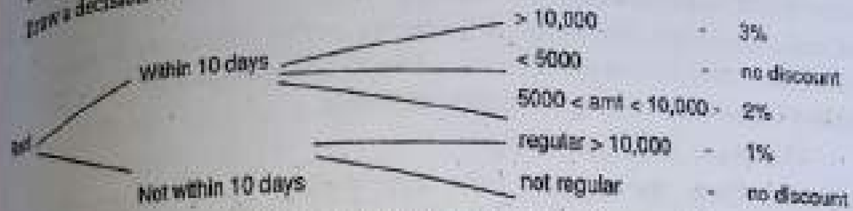


Fig. 5.4: Decision tree of company on discounts

Advantages of Decision Tree:

1. Decision trees are simple to understand and interpret.
2. It helps the analyst to identify the actual decision to be made.
3. It is used to verify logic and problems that involve a few complex decisions and limited number of actions.
4. Decisions trees can be combined with other decision techniques.

Disadvantages of Decision Tree:

1. A large number of branches with many paths will confuse rather than help in analysis.
2. Large decision trees can be hard to interpret.
3. Decision tree uses a step function that can have large errors near boundaries.

5.2.2 Decision Table

In a table form

BCA-S-18, W-18

- A decision table is a graphical method for explaining the logic of making decisions in tabular format.
- A decision table is a matrix representation of logic of decisions that specify the possible conditions for decision and resulting actions.
- A Decision Table (DT) is defined as, "a tabular representation used to describe and analyze decision situations, where the state of a number of conditions determines the execution of a set of actions".

Use a Decision Table (DT):

1. To show sets of conditions and the actions resulting from them when the logic can be easily expressed in a table format, for example, calculating discount rates.
2. To verify completeness and consistency of a process involving different actions under different conditions.

- A typical decision table is divided into four parts as shown in Fig. 5.4.

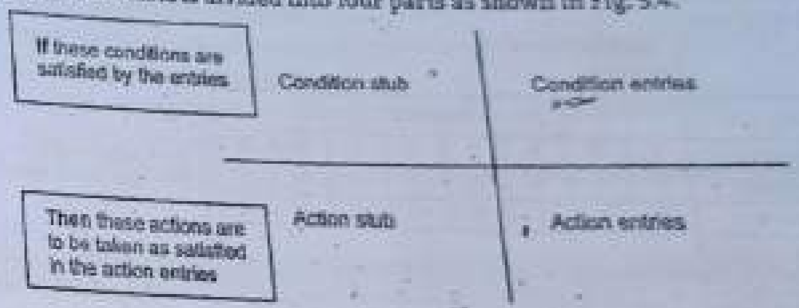


Fig. 5.4: Parts of Decision Table

- Fig. 5.4 shows the following parts of DT:
  1. **Condition stub (statements)** which shows the conditions that determine which actions will result.
  2. **Condition entries** are the combination of conditions expressed as rules.
  3. **Action stub (statements)** which contains the possible actions which can occur as a result of the different condition combinations.
  4. **Actions entries** which contains the action to be taken.

- In the right part of a decision table, each column is named by a rule number or rule identifier.

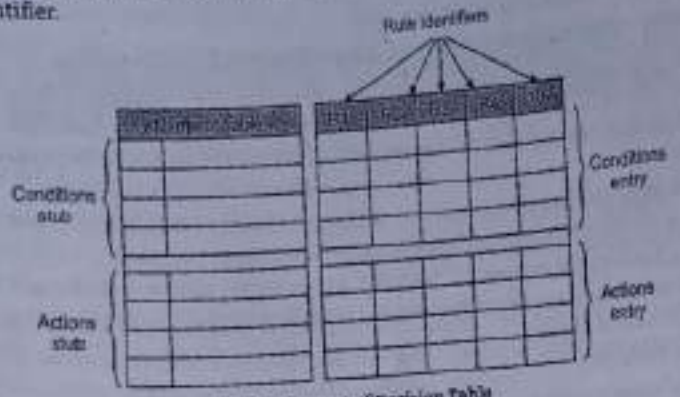


Fig. 5.5: Format of Decision Table

Basic Idea behind Decision Table:

- Fig. 5.6 shows a decision table. In a nominal limited entry decision table, a decision indicator is just a Y (Yes) or N (No) to indicate that the given condition must be fulfilled or not. An indicator of action is an "X" for those actions to execute and "-" for those actions to drop.

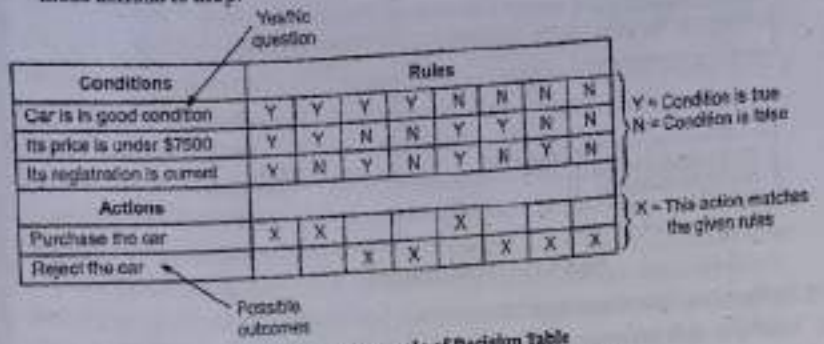


Fig. 5.6: Example of Decision Table

- Various parts of the decision table in Fig. 5.6 are discussed below:
  1. **Conditions:** These are created from each decision question. Only one possible answer from each question is selected for use in the table. Each answer corresponds to one of each pair of branches in the tree.
  2. **Actions:** These are the final outcomes of the decision process and are the branch ends or outcomes of the decision tree.

3. **Rules:** These give the combinations of conditions that lead to the final actions. Y (for Yes) and N (for No) characters in the table normally indicate which combinations of conditions are allowed.

Advantages of Decision Table:

1. A decision table can be changed according to situation.
2. A decision table provides a framework for a complete and accurate statement of processing or decision logic.
3. A decision table may be easier to construct than a flowchart.
4. A decision table is compact and easily understood making it very effective for communication between analysts and non-technical users.
5. It provides compact representation of decision making process.
6. The structure of decision table promotes a logically complete and consistent problem definition.

Disadvantages of Decision Table:

1. A Decision Table does not show the flow of logic for the solution to a given problem.
2. If there are too many alternatives, it is difficult to list in decision table.
3. It cannot express the complete sequence of operations to solve a problem therefore it may be difficult for the programmer to translate decision table into program.

Table Entries in Decision Tables:

- Types of table entries in decision tables are explained below:
  1. **Limited Entry Form:** The table structure consisting only of Y, N and blank entries in the condition entry section and X in the action entry section is a limited entry form table. It is one of the most commonly used format.

     Volume ⇒ Total Transaction

  2. **Extended Entry Form:** The extended entry form replaces Y and N with actual entries. In this format, the condition and action statements themselves are not complete. Therefore the entry sections contained more details than Y or N.

     For example:

Condition	Decision Rules						
Time	Within 10 days	Within 10 days	Within 10 days	Not within 10 days	Not within 10 days		Not within 10 days

(cont.)

Business volume	Over 10,000	Between 5000 and 10,000	Below 5000	Over 10,000	Between 5000 and 10,000	Below 5000
Action	3%	2% discount	No discount	No	No	No

3. **Mixed Entry Form :** Analyst may combine the features of both the above form in the same table

Condition	Decision Rules			
Time	Within 10 days	Within 10 days	Within 10 days	
Business volume	Over 10,000	Between 5000 and 10,000	Below 5000	
3% discount	X			
2% discount		X		
No discount			N	

4. **Else Form:** This form is used to avoid repetition in table. To build an else form, decision table analyst need to specify the rules with condition entries to cover all set of actions except for one, which will be the rule to follow when none of the other conditions is true. This rule is in the final column on the right of the table.

For example:

Condition	Decision Rules		
Time	Within 10 days	Within 10 days	Else
Business volume	Over 10,000	Between 5000 and 10,000	Else
3% discount	X		
2% discount		X	
No discount			N

5.2.1 Examples for Decision Table (BCA: S-19)

- **Example 1:** An insurance company uses the following rule to determine the eligibility of a driver for insurance. The driver will be insured if:

1. The driver lives in the city with population less than 5000 and he is a married man.
2. The driver lives in the city with population less than 5000 and he is a married and age is over 30 years old.
3. The driver lives in the city with population is 5000 or more and it is married female.
4. The driver is male over 30.

5. The driver is married and under 30.

Write Decision Table for above cases.

Solution:

Condition	Rule1	Rule2	Rule3	Rule4	Rule5	Rule6
Lives in city population<5000	Y	Y	N	-	-	-
Male	Y	-	N	Y	-	-
Married	Y	Y	Y	-	Y	-
Age>30	-	Y	-	Y	N	-
Action	**Rule1**	**Rule2**	**Rule3**	**Rule4**	**Rule5**	**Rule6**
Insured	Y	Y	Y	Y	Y	-
Not insured						Y

- **Example 2:** Study following conditions and draw a decision table:

1. If product code=A and customer type=1 and the order amount<=700
   Then 5% discount allowed
2. If product code=A and customer type=2 and the order amount <=700
   Then 7.5% discount allowed
3. If product code=A and customer type=1 and the order amount >=700
   Then 7.5% discount allowed
4. If product code=A and customer type=2 and the order amount >700
   Then 10% discount allowed
5. A flat discount of 5% on product code=B regardless of customer type and the order amount.

Solution:

Condition	Rule1	Rule2	Rule3	Rule4	Rule5	Rule6
Product code=A	Y	Y	Y	Y	N	-
Customer type=1	Y	N	Y	N	-	-
Order amount<=700	Y	Y	N	N	-	-
Action	**Rule1**	**Rule2**	**Rule3**	**Rule4**	**Rule5**	**Rule6**
Discount 5%	Y				Y	
Discount 7.5%		Y	Y			
Discount 10%				Y		
No Discount						Y

- **Example 3:** A Co-operating bank XYZ granted loan under following conditions:
  1. If a customer has an account with the bank and has no loan outstanding (no dues), loan will be granted.
  2. If a customer has an account is outstanding from previous loan, loan will be granted if special management approval is obtained.
  3. Reject loan application in all other cases.

Draw decision table for above cases:

Solution:

Condition	Rule1	Rule2	Else
Customer has Account	Y	Y	-
No Outstanding loan amount	Y	N	-
Special Management Approval	-	Y	-
Action	**Rule1**	**Rule2**	**Else**
Loan Application Accept	Y	Y	
Loan Application Reject			

5.3 DATA FLOW DIAGRAMS (BCA: S-18, W-18)

- IEEE defines Data Flow Diagram (DFD) as, "a diagram that depicts data sources, data storage, and processes performed on data as nodes, and logical flow of data as links between the nodes."
- The Data Flow Diagram is a graphical representation of the flow of data through an information system.
- A data-flow diagram also known as Bubble Chart or Work Flow Diagram.
- A DFD maps out the flow of information for any process or system. It uses defined symbols like rectangles, circles and arrows, to show data inputs, outputs, storage points and the routes between each destination.

Uses:

- DFD enables us to represent the processes in the information system from the viewpoint of data.
- The DFD lets us to visualize how the system operates, what the system accomplishes and how it will be implemented, when it is refined with further specification.
- Data flow diagrams are used by systems analysts to design information-processing whole organizations.

Characteristics:

- DFD accomplishes the following characteristics:
  1. A DFD represents system data in a hierarchical manner and with required level of detail.
  2. A DFD shows processes according to defined user requirements and software scope.
  3. DFD focus on the process that transforms incoming data flows (input) to outgoing data flows (output).

5.3.1 Symbols used in DFD

- Following table shows symbols used in DFDs with example.

Name	Symbol	Description	Example
Entity		A system comes into action based on input signal or data it receives from an entity. Used to represent people and organizations outside the system. They either input information to the system; accept output information from the system or both. This symbol also known as External entity.	Customer
Process		The process causes some transformation to data. These are actions that are carried out with the data that flows around the system. A process accepts input data and produces data that it passes on to another part of the DFD.	Verify order
Data Flow		A data element that goes into a process as input or out of a process as output is called as data flow. These represent the flow of data to or from a process.	Customer details

Data store		This is a place where data is stored either temporarily or permanently. Data elements stored in a data file are called Data Store.	Products

- Fig. 5.7 shows Basic Notations in DFDs.

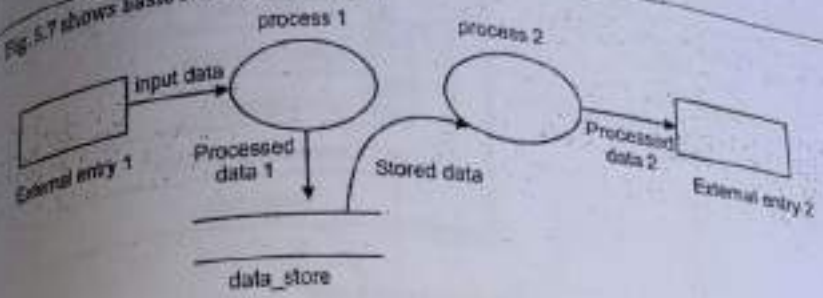


Fig. 5.7: Basic Notations of DFD

5.3.2 Types of DFD

[BCA:W-18]

- There are two types of DFDs, both of which support a top-down approach to systems analysis as given below:

1. **Logical DFDs** are implementation-independent and describe the system, rather than how activities are accomplished. The logical DFD specifies the various logical processes performed on data i.e., type of operations performed.
2. **Physical DFDs** shows how the system will be implemented. A physical DFD specifies who does the operations whether it is done manually or with a computer and also where it is done.

- Physical DFD to issue a book from library is shown in Fig. 5.8.

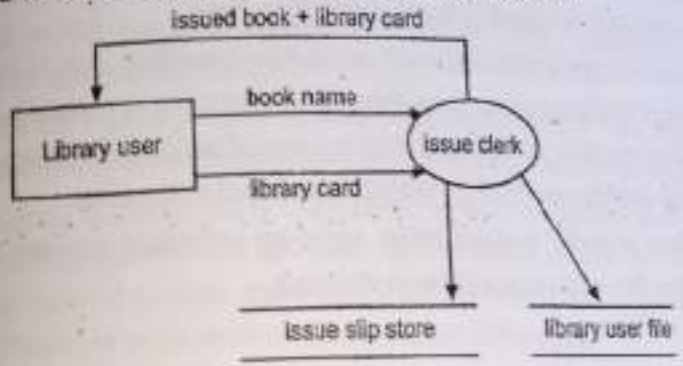


Fig. 5.8: Physical DFD

- Logical DFD to issue a book from library in Fig. 5.8 as shown in Fig. 5.9.

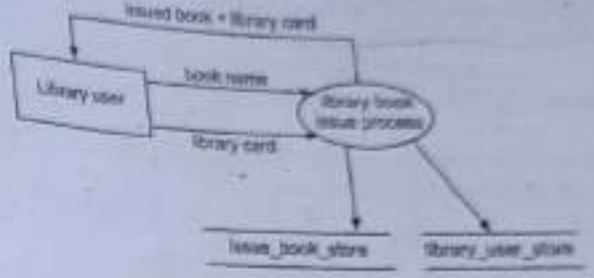


Fig. 5.9: Logical DFD

- Here, library user will give book name and library card, after successful issue his name and book name will be stored in issue_book store and library_user_store contain all the books issued to the user.

5.3.3 Levels of DFD

[BCA:S-19]

- DFD supports a top-down approach for analysis. There are various levels of DFD, which provide details about the input, processes, and output of a system.
- Note that the level of detail of process increases with increase in level(s).
- However, these levels do not describe the system's internal structure or behavior. These levels are listed below:

1. **Level 0 DFD:** This shows an overall view of the system. Level a DFD is also known as context diagram.
2. **Level 1 DFD:** This expands level 0 DFD and splits the process into a detailed form.
3. **Level 2 DFD:** This expands level 1 DFD and displays the process(s) in a detailed form.
4. **Level 3 DFD:** This expands level 2 DFD and displays the process(s) in a detailed form.

- Levelled DFD are drawn in top-down approach, i.e. at top level 0 level DFD or context is made then this 0 level DFD is expanded to level 1 DFD from (expanding and dividing) the main process bubble of 0 level DFD.
- At level 1, the process bubbles are also called as the sub-processes. To draw level 2 DFD, these sub-processes are further expanded and divided individually in 3 to 7 bubbles and so on. A detailed illustration of levelling is shown in Fig. 5.10.

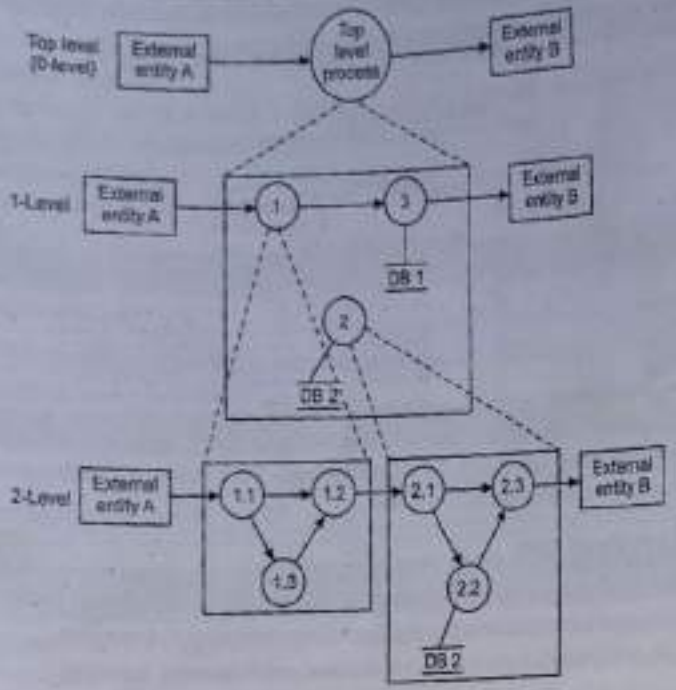


Fig. 5.10: Levelled DFD

Examples of Levelled DFD:

- To understand various levels of DFD, let us see various examples as follows:

Example 1: Levelled DFD for Banking System.

- **Level 0 DFD:** Level 0 DFD (also known as the context level DFD) is the simplest DFD.
  - The outermost level (level 0) is concerned with how the system interacts with the outside world. This level basically represents the input and output of the entire system.
  - The Level 0 DFD depicts the entire Banking system as a single process. There are various tasks performed in a Bank such as Transaction processing, Passbook entry, Registration, Demand draft creation, Online help and so on.
  - The data-flow indicates that these tasks are performed by both the user (customer) and the bank. When the user performs a transaction, the bank verifies whether the user is registered in the Bank.

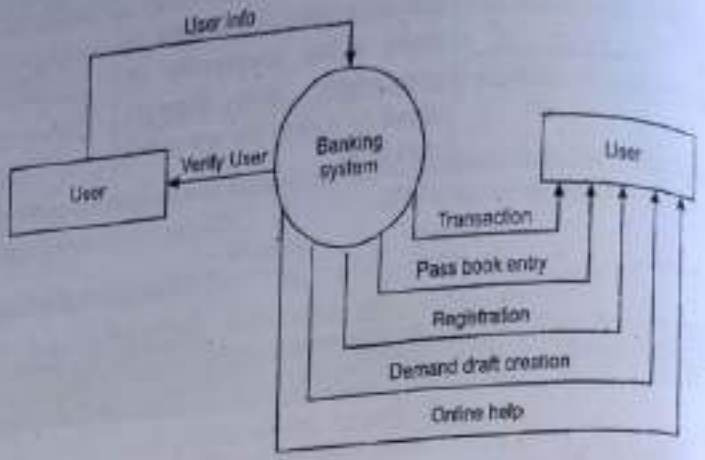


Fig. 5.11: Level 0 DFD

- **Level 1 DFD:** The Level 0 DFD is expanded in Level 1 DFD.
  - In Level 1 DFD, the 'User' entity is related to several processes in the Bank, which include 'Register', 'User support', and 'Provide cash'.
  - Transaction can be performed only if the user is already registered in the Bank. Once, the user is registered, he can perform a transaction by the process namely, 'Deposit cheque', 'Deposit cash' and 'Withdraw cash'.
  - If the user is performing transaction 'Deposit cheque', the user needs to give cheque to the bank. The user's information such as name, address, and the account number is stored in 'user-detail' data store, which is a database.
  - If cash is to be deposited and withdrawn, then the information about the deposited cash is stored in 'Cash-detail' data store.
  - The user can get a demand draft created by providing cash to the bank. It is not necessary for the user to be registered in that bank to have a demand draft. The details of amount of cash and date are stored in 'DD-detail' data store. Once the demand draft is prepared its receipt is provided to the user.
  - The 'User support' process helps users by providing answers to their queries related to the services available in the bank.

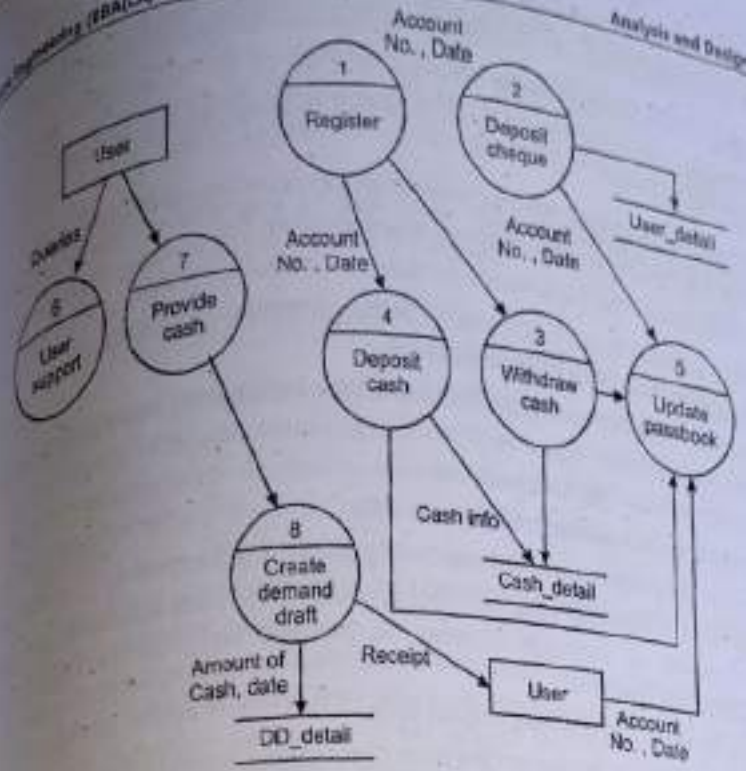
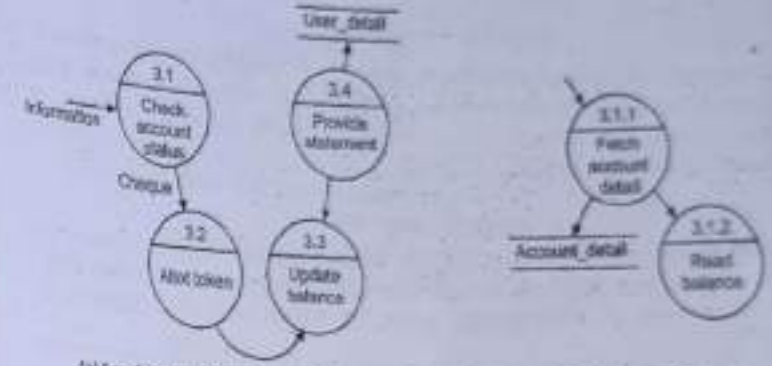


Fig. 5.12: Level 1 DFD

- **Level 2 DFD:** Level 1 DFD can be further refined into Level 2 DFD for any process of a banking system that has detailed tasks to perform.
- For instance, Level 2 DFD can be prepared to deposit a cheque, deposit cash, withdraw cash, provide user support, and to create a demand draft. However, it is important to maintain the continuity of information between the previous levels (Level 0 and Level 1) and Level 2 DFD.
- As mentioned earlier, the DFD is refined until each process performs a simple function, which is easy to implement.
- Let us consider the 'Withdraw cash' process to illustrate Level 2 DFD, (See Fig. 5.13 (a)). The information collected from Level 1 DFD acts as an input to Level 2 DFD.
- To withdraw cash, the bank checks the status of balance in the user's account ('Check account status' process) and then allots a token ('Allot token' process). After the user withdraws cash, the balance in user's account is updated in the 'User_detail' data store and a statement is provided to the user.

- **Level 3 DFD:** If a particular process of Level 2 DFD requires elaboration, then this level is further refined into Level 3 DFD.
- Let us consider the process 'Check account status' to illustrate Level 3 DFD. (See Fig. 5.13 (b)). To check the account status, the bank fetches the account detail ('Fetch account detail') from the 'Account_detail' data store.
- After fetching the details, the balance is read ('Read balance') from the user's account.
- Note that the requirements engineering process of DFDs continues until each process performs a function that can be easily implemented as an individual program component.



(a) Level 2 DFD to Withdraw Cash (b) Level 3 DFD to Check Account Status

Fig. 5.13

Example 2: Levelled DFD for Online Shopping System.

- **Level 0:** Highest abstraction level DFD is known as Level 0 DFD, which depicts the entire information system as one diagram hiding all the underlying details. Level 0 DFDs are also known as context level DFDs.

Fig. 5.14: Level 0 DFD

- **Level 1:** The Level 0 DFD is broken down into more specific Level 1 DFD. Level 1 DFD depicts basic modules in the system and flow of data among various modules. Level 1 DFD also mentions basic processes and sources of information.

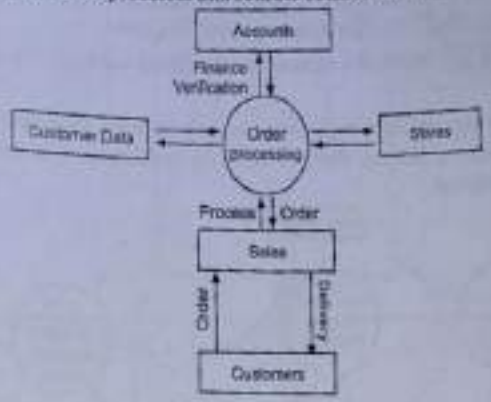


Fig. 5.15: Level 1 DFD

- **Level 2:** At this level, DFD shows how data flows inside the modules mentioned in Level 1. Higher level DFDs can be transformed into more specific lower level DFDs with deeper level of understanding unless the desired level of specification is achieved.

Example 3: DFDs for Book Ordering System

- DFD for a Book Ordering System as shown in Fig. 5.16.

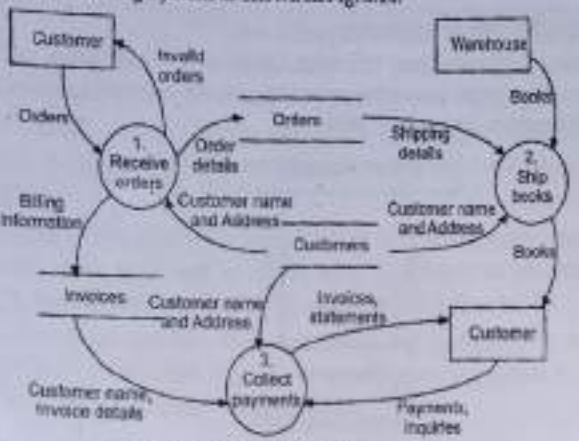


Fig. 5.16: DFD for a Book Ordering System

Advantages of Data Flow Diagrams (DFDs):

1. A simple graphical technique which is easy to understand by technical and non-technical users.
2. It is useful for communicating current system knowledge to the users.
3. A DFD is used as the part of system documentation file.
4. A DFD explains the logic behind the data flow within the system.
5. DFDs can provide a detailed representation of system components.

Disadvantages of Data Flow Diagrams (DFDs):

1. Data flow diagram undergoes lot of alteration before going to users, so makes the process little slow.
2. The DFD takes a long time to create.
3. Physical considerations are left out in DFD.
4. It make the programmers little confusing concerning the system.
5. Different DFD models have different symbols like in Gane and Sarson process is represented as rectangle whereas in DeMarco and Yourdan symbol it is represented as eclipse.

Comparison between Physical and Logical DFDs:

Table 5.2: Difference between Physical and Logical DFDs

Sr. No.	Physical DFD	Logical DFD
1.	The DFD that shows "what is going on" is a physical DFD.	The DFD that shows "how it is going on" is a logical DFD.
2.	Physical DFDs are more detailed in nature.	Logical DFDs are more abstract in nature.
3.	These DFDs are more realistic and implementation oriented.	Logical DFDs are more logical in format.
4.	Physical DFDs shows physical component and system like people, documents, reports etc.	Logical DFDs shows work done without referring to order processing etc.
5.	Physical DFD shows an implementation dependent view of the current system.	Logical DFD shows an implementation independent view of the current system.
6.	Physical DFDs just for visualization.	Logical DFD help to get a clear idea of what system is achieving.

5.4 DATA DICTIONARY

BCA: S-18, S-19

- A Data Dictionary (DD) stores an organized collection of information about data and their relationships, data-flows, data types, data stores, processes and so on.
- In addition, it helps users to understand the data types and processes defined along with their uses.
- It also facilitates the validation of data by avoiding duplication of entries and provides the users with an online access to definitions.

Definition:

- A data dictionary refers to, "a collection of descriptions of the data objects or items in a system". OR
- A data dictionary can be defined as, "a centralized repository of information about system and its elements such as meaning, relationships, origin, usage and format".
- Although the format of data dictionaries varies from method to method, most use the (see Fig. 5.17) called as project.

Types of DD:

- Based on the usage of a data dictionary by a particular entity, it may be seen under two categories as described below.

1. **Passive Dictionary:** If a data dictionary is being used by the users, designers and the database administrators then the data dictionary is called as passive dictionary. Basically, a passive dictionary is used for the purpose of software documentation only. It enables a developer to maintain a separate database, which is independent and is in the state of non-synchronization to a particular relational database management system.
2. **Active Dictionary:** Data dictionary being directly used and managed by the Database Management System Software is known as active dictionary. This dictionary is consistent and in synchronization to the relational database management system.

5.4.1 Elements of DD

BCA: W-18

- The data dictionary has following types of components or elements:

1. **Data Element:** Data element is the fundamental level or the elementary item. Data elements are building blocks for all other data in the system. It is a smallest unit of data which cannot be meaningfully decomposed. For example, Invoice number, invoice date and amount due are the data elements.
2. **Data Structure:** Data structure is a group of data elements handled as a unit. Data elements are grouped together to makeup a data structure. A data structure is a set of data items that are related to one another and collectively describe a component in a

system. Data structure shows the information about the formation of the data item. Fig. 5.21 illustrates this. For example, Phone is a data structure consisting of four data elements: area code-exchange-number-extension.

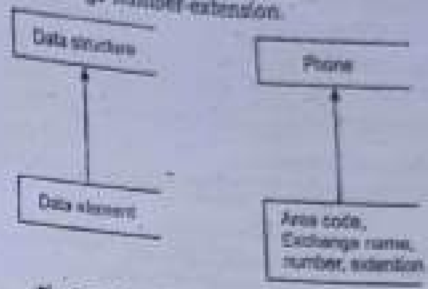


Fig. 5.17: Data Structure and Data Element

3. **Data Flows and Data Stores:** It includes the complete explanation of all the elements used in data flow diagram and a description of each data flow, data structure and process. Data flows are data structures in motion, whereas data stores are data structures at rest. A data store is a location where data structures are temporarily located. Fig. 5.22 shows data flows and data stores.

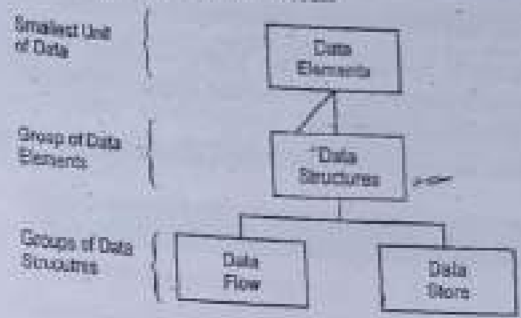


Fig. 5.18: Data flows and Data stores

- Data element sheet and information about the data item are shown in Fig. 5.19.

[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

Fig. 5.19: Data element sheet

- For example, an example of integrated office phone system. (See Fig. 5.20).

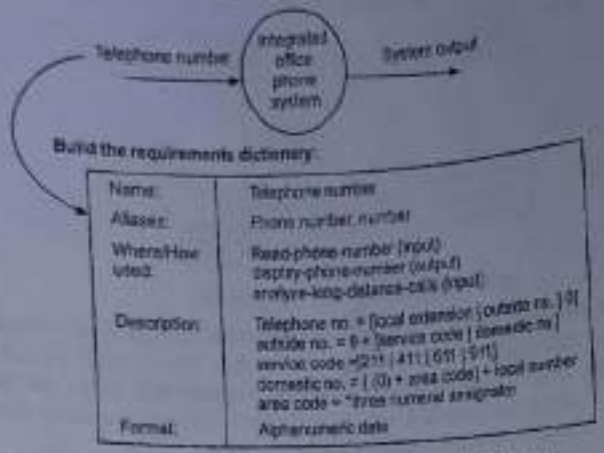


Fig. 5.20: Data Dictionary of Integrated office phone system

- The description format is frequently given a highly structured form but it is to be noted that consistency in naming must be followed. Except the content description all the information may be provided directly.
- To develop the content description of a data item some notations (symbols) as shown in Fig. 5.21.

Data Construct	Symbol	Meaning
	=	is composed of/equivalent to
Sequence	+	and
Selection	[|]	either-or
Repetition	{ }^{n}	n repetitions of/iterations of
	( )	optional data entry
	* _ *	delimits comments

Fig. 5.21: Notations of Data Item

- Above notation enables representation of composite data in one of the three fundamental ways that it can be constructed:

- As a sequence of data items.
- As a selection from among a set of data items.
- As a repeated grouping of data items. Each data item entry that is [illegible] part of a sequence, selection, or repetition may itself be another [illegible] item that needs further refinement within the dictionary.

4. **Data Names:** A formal or primary name of the data or control item [illegible] external entity. These names given for data elements should be [illegible] others like Name, Age, Sex etc.
5. **Data Description:** It shows in detail what the data item represents in the [illegible] is a short description of data.
6. **Alias:** Other names or acronyms of data item are called as aliases. For [illegible] can be referred as SSNO or SS_NO etc.
7. **Use:** A listing of processes which use the data or control item and how it [illegible] used.
8. **Data Type:** It shows type of data like Numeric, Float, etc.
9. **Length:** Length means number of spaces required to store the data element. For example, stud_ID number 4 it stores the numbers from 0 to 9999.
10. **Data Value:** In some processes only specific data values are allowed, [illegible] additional prefixes are added called as data values.

For example, SYBCA001; Second Year of Bachelor in Computer Application Roll No. SY 001: Second Year BCA

5.4.2 Advantages and Disadvantages of DD

Advantages:

1. A DD manage the details in large systems and DD improves consistency.
2. It is used to locate omissions and errors in the system.
3. A DD is used to communicate common meaning for all system elements.
4. It is a valuable reference in any organisation.
5. It improves analyst/user communication by establishing consistent [illegible] various elements, terms and procedures.

Disadvantages:

1. For large organization, a DD is grows rapidly in size and complexity.
2. A DD is difficult to maintain manually.
3. Some users and analysts who may use other methods find it difficult [illegible] their trade and start to use DDs.

5.5 INPUT AND OUTPUT DESIGN

- Software design is a process to conceptualize the software requirements into software implementation.
- Software design takes the user requirements as challenges and tries to find optimum solutions. While the software is being conceptualized, a plan is written to find the best possible design for implementing the proposed solution.

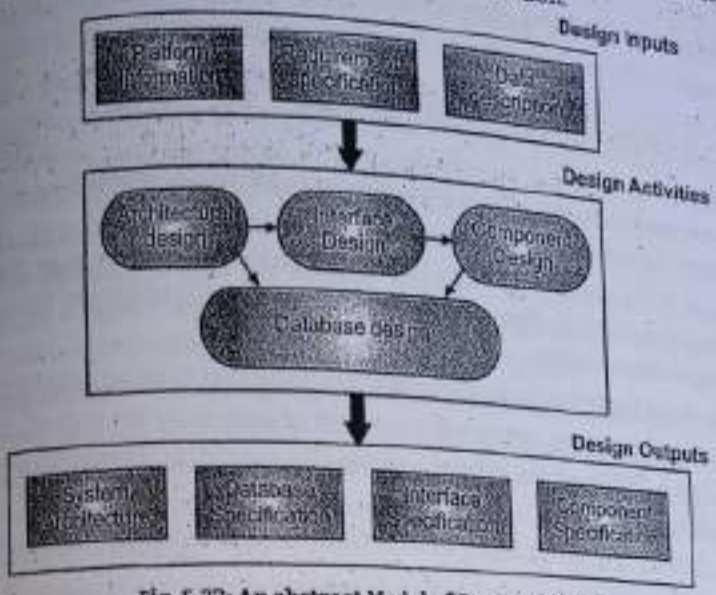


Fig. 5.22: An abstract Model of Design Process

- A software design is a description of the structure of the software to be implemented, the data models and structures used by the system, the interfaces between system components and, sometimes, the algorithms used.
- Designers do not reach at a finished design immediately but develop the design iteratively. They add formality and detail as they develop their design with constant backtracking to correct earlier designs.
- Fig. 5.23 is an abstract model of design process showing the inputs to the design process, process activities, and outputs from the process. In this fig., stages of the design process are sequential. In fact, design process activities are interleaved. Feedback from one stage to another and consequent design re-work is expected in all design processes.

5.5.1 Input Design

(BCA: S-19)

- Input design means designing the screens used to enter the information, as well as any forms on which users write or type information like Time Cards.

- Input provides facilities the entry of data into the computer system. Input design involves the selection of the best strategy for getting data into the computer system at the right time and as accurately as possible.
- The process of input design starts with the design of the source documents that capture the data. This is followed by the selection of appropriate media to enter the same into the computer.
- The goal of input design is to capture accurate information for the system simply and easily.
- The fundamental principles for input design reflect the nature of the inputs (whether batch or online) and ways to simplify their collection.
- The system analysts should work out the following input design details:
  1. What data is to be input?
  2. What medium is to be used?
  3. How should the data be arranged or coded?
  4. The interface to be provided to the users for providing input.
  5. Intities and transactions requiring validation to detect errors.
  6. Ways to perform input validation and measures to take in case error occurs.

Objectives of Input Design:

- Input design consists of developing specification and procedures for data preparation, and those steps necessary to put transaction data into a usable form for data entry and processing.
- In other words, input design is the activity of putting data into the computer for processing.
- The objectives of input design are given below:
  1. **Controlling amount of Input:** Because data entry operations are dependent of people, if possible less data entry leads low cost. Second as data input is high leads much time consumption for processing.
  2. **Avoiding Delay:** This means how analyst can design input of the system so that time delay can be reduced for input.
  3. **Avoiding Errors in Data:** The rate of errors directly depends on data input. The analyst can also affect the error rates of an operation through input design by the way data be entered.
  4. **Avoiding Extra Steps:** When the volume of transactions can not be reduced, the analyst must be sure the process is as efficient as possible. Perfect analyst will also avoid input designs that cause extra steps. This will save time as well less data entry.

5. **Keeping the Process Simple:** The users always accept simple input design. It is advisable to avoid complexity when there are simple alternatives for complex systems.

- The simple input layout (for computer screen) is given below:

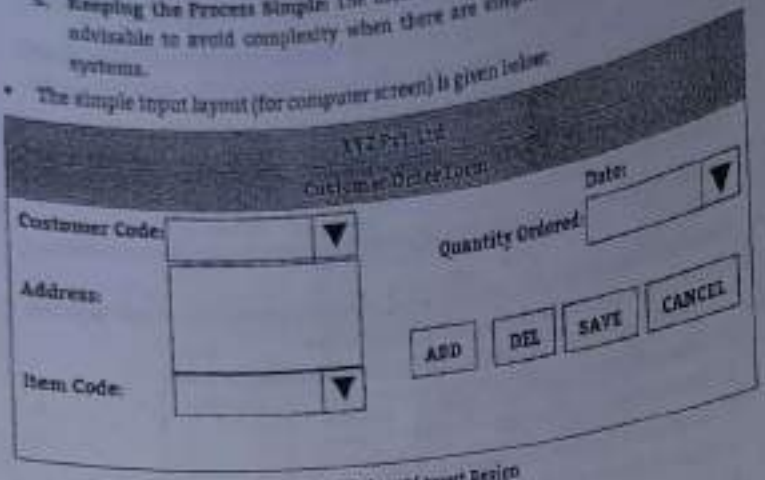


Fig: 5.23: Input Layout Design

5.5.1.1 Common GUI Controls for Inputs

- **Text Box:** It can allow for single or multiple lines of characters to be entered.
- **Radio Button:** Radio buttons provide the user with an easy way to quickly identify and select a particular value from a value set. A radio button consists of a small circle and an associated textual description those responses to the value choice. Radio buttons normally appear in groups as one radio buttons per value choice.
- **Check Box:** It consists of a square box followed by a textual description of the input field for which the user is to provide the Yes/No value.
- **List Box:** A list box is a control that requires the user to select a data item's value from the list of possible choices. It is rectangular and contains one or more rows of possible data values. The values may appear as either a textual description or graphical representation.
- **Dropdown List:** It is like a list box, but is intended to suggest the existence of hidden list of possible values for a data item.
- **Combination Box:** It is also known as combo box. It combines the capabilities of a text box and list box. A combo box gives the user, the flexibility of entering a data item's value or selecting its value from a list.
- **Spin Box:** A spin box is used to allow the user to make an input selection by using the buttons to navigate through a small set of meaningful choices.

5.5.1.2 Process of Input Design

- The following steps are to be followed during the process of input design are below:
  1. Identify system inputs and review logical requirements.
  2. Select appropriate GUI (Graphical User Interface) controls.
  3. Design, validate and test inputs using some combination of
     i) Layout tools (e.g., hand sketches, printer/display layout chart, or Case)
     ii) Prototyping tools (e.g., spreadsheet, PC DBMS, 4GL)
  4. If necessary, design the source document.

5.5.1.3 Basic Terms used in Input Design

1. **Data Capture:**

- Data capture covers all stages from recording of basic data to the feeding of it into computer for processing.
- Data capture is the capture the data from the outside world other than the system which has been developed. There are general guidelines that will assist the analyst in formulating an input design.
- The analyst should start by capturing only those items that must actually be input.
- Data capture input of data, not as a direct result of data entry but instead as a result of performing a different but related activity like capturing inventory record while recording a sale.
- It is an identification and extraction of data from a scanned document.

Objectives of Data Capture:

(i) Reduction in volume of input output to the extent possible.
(ii) Lesser manual preparation.
(iii) An input design will ease the work of the person engaged in input preparation.
(iv) Minimize the number of steps practicable in data capturing process.

- There are two types of data that must be input when processing transactions.
  (i) **Variable Data:** Those data items that change for each transaction based decision made.
  (ii) **Identification Data:** The element of data that uniquely identifies the item be processed.

2. **Data Entry:**

- It is the process of translating the source document into the machine sensible format.

(i) **Constant data:** Data that are the same for every entry. For example, since the date of the transaction is identical for every transaction originating on specific data, it need not be input for every transaction.

(ii) **Details that System can retrieve:** Data that can be easily accessed from system files need not be entered. For example, when entering inventory data, using the identifying item number, the data entry personnel should not be required to provide the item description.

(iii) **Details that the System can compute:** Desired results that can be obtained by using the combination of stored and entered data. For example, to calculate the cost of items sold, the system should only require the operator to provide the item number and quantity purchased. The input data design process starts by the design of the source documents that capture the data.

(iv) **Source Document:** Data are initially captured on the source document. For example, a bank cheque used for financial transactions is a source document that starts a processing cycle when entered into a system.

(v) **Coding Method:** While designing information systems, due consideration should be given to space, time and cost. Coding method add to the design efficiency by saving space, time and cost. A code is a brief number, title, or symbol used instead of more lengthy or ambiguous description. Various coding methods are adopted for these are:

(a) **Function Codes:** Function codes state the activities or work to be performed without spelling out all of the details in narrative statements. User uses this type of coding method to process the data.

(b) **Sequence Codes:** Sequence codes are either numbers or letters assigned in series.

(c) **Mnemonic Codes:** Mnemonic codes use letters and symbols from the product to describe it in a way that communicates visually.

Identifying Data Entry Requirements and System Responsibilities:

- In Fig. 5.24, to compute the cost of items sold, the system can require the operator to provide the item number and quantity purchased.
- Then the design specifies that the system will retrieve the unit price and multiply it by the quantity to produce the cost extension for one item. It also totals all the extensions for an order, calculates sales tax and draws a grand total.
- Only a few items need be entered for this processing to occur. Hence, data entry is the data that the user has to enter to complete the transactions.

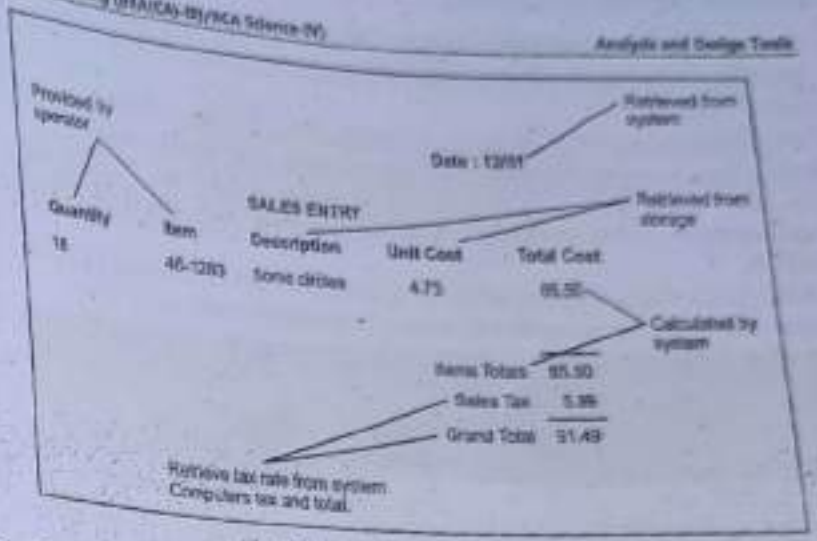


Fig. 5.24: Required fields in data entry

3. **Data Input:**

- To input data in the existing way of format, provided by the system.
- The goal of designing input data is to make data entry as easy, logical and free from errors as possible.
- While entering data, operators need to know the following things:
  (i) The allocated space for each field.
  (ii) Field sequence, which must match that in the source document.
  (iii) The format in which data fields are entered.
- For example, Filling out the date field is required through the edited format mm/dd/yy.

5.5.2 Output Design

- Useful outputs are the final products of a good information system. Therefore, the design of system outputs marks the beginning of a good system design work.
- Output design involves specifying how production of on-screen reports and paper-based reports will occur.
- Outputs can be classified according to two characteristics: i) Their distribution inside or outside the organization and the people who read and use them; and ii) Their implementation method.
- In output design, importance is given on generating a hard copy or a soft copy of the information requested in a predefined format. Understanding the type and purpose of the output is also essential.

- The goal of the output mechanism is to present information to users so that they can accurately understand it with the least effort.

Formats of Output:

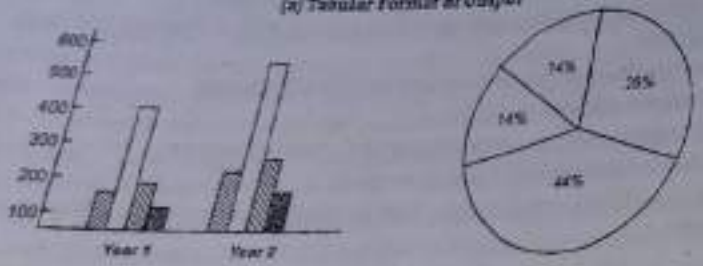
- Commonly used formats of output are: Tabular format and Graphical format.
- Tabular format presents information as rows and columns of text and numbers.
- Graphical format is the use of a picture to convey information. Its ways that demonstrate trends and relationships. It may be using a bar chart, a pie chart, an area chart etc.
- Following table displays tabular and graphic forms of output. Tabular and graphical formats may be combined together to improve the presentation of output.

XYZ Pvt. Ltd.

Purchase Order

Sl. No.	Item Code	Quantity Ordered
1.		
2.		
3.		

(a) Tabular Format of Output



(b) Graphical Format of Output

Fig. 5.25

Objectives of Output Design:

1. Designing output to serve the intended purpose.
2. Designing output to fit the user.
3. Delivering the appropriate quantity of output.
4. Making sure the output is where it is needed.
5. Providing the output on time.
6. Choosing the right output method.

5.5.2.1 Types of Outputs

- The users of output generated by a computer can be varied. For example, computer manufacturer, a computer using firm is a user. To the computer section of an organization, anyone external to the section who receives output the computer.
- Output can be categorized as follows:

1. **External Outputs:** These outputs are received by entities outside the organization. The invoices, orders, etc. Special attention is given to the format, content organization of these kinds of outputs as they directly have a bearing on organization's business relations with its customers.
2. **Internal Outputs:** These outputs are received by people within the organization and require careful consideration because they may affect the operational system.
3. **Computer Operations Outputs:** These outputs are used within the computer services department such as usage statistics, control reports, etc.
4. **Interactive Outputs:** It involves the user communicating directly with the computer.
5. **Turnaround Outputs:** In this, the data will be added to the document before

5.5.2.2 Process of Output Design

- The steps to be followed during process of output design are given below:
  - i) Identify system outputs and review logical requirements.
  - ii) Specify physical output requirements.
  - iii) As necessary, design any pre-printed external forms.
  - iv) Design, validate and test outputs using some combination of:
    - a. Layout tools (e.g., hand sketches, printer/display layout chart, or CASE).
    - b. Prototyping tools (e.g., spreadsheet, PC DBMS, 4GL).
    - c. Code generating tools (e.g., report writer).

5.6 PSEUDOCODE

- Pseudocode is made up of two words 'pseudo and code'. Pseudo means imitation or false and code refers to instructions written in a programming language.
- Pseudocode is simply a way to communicate specifications to programmers or others. Pseudocode is a compact and informal high-level description of a computer programming algorithm that uses the structural conventions of some programming language.
- Pseudocode notation can be used in both the preliminary and detailed design phases in a system.
- Using pseudocode, the designer describes system characteristics using short, concise English language phrases that are structured by keywords such as IF-THEN-ELSE, WHILE-DO and END.

- Keywords and indentation describe the flow of control, while the English phrases describe processing actions.
- Pseudocode is also known as Program Design Language (PDL) or structured English.

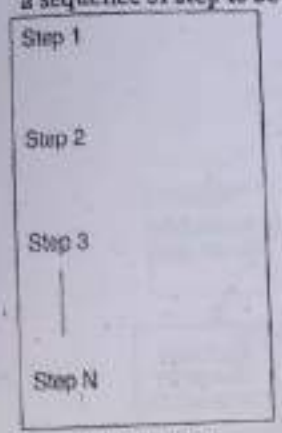
Characteristics:

- A program design language should have the following characteristics:
  1. A fixed syntax of keywords that provide for all structured constructs, data declarations and modularity characteristics.
  2. A free syntax of natural language that describes processing features.
  3. A data declaration facility.
  4. Sub-program definition and calling techniques.

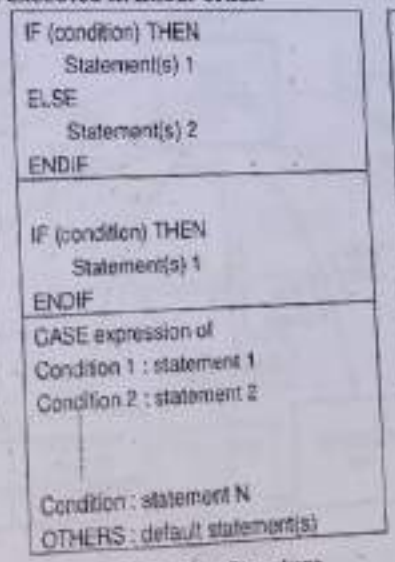
```
BEGIN Fees Balance
   /* Comment: Checking student fees balance */
   FOR EACH student
      Retrieve student_Reg_No, Amount_paid
      IF Amount_paid is less tan Fees_Amount,
         THEN Send_Reminder
      ENDIF
   NEXT student
END
```

Structures in Pseudocode:

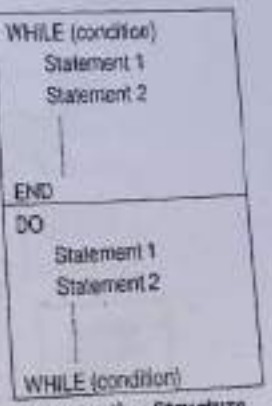
- Fig. 5.26 shows the different pseudocode structures. The sequence structure is simply a sequence of step to be executed in linear order.



(a) Sequence Structure



(b) Selection Structure



(c) Iteration Structure

Fig. 5.26: Control Structures for Pseudocode

- There are two main selection constructs IF statement and CASE statement.
- In the IF statement, if the condition is true then the THEN part is executed otherwise the ELSE part is executed.
- The CASE statement is used where there are a number of conditions to be checked.
- WHILE and DO-WHILE are the two iterative statements. The WHILE loop and the DO-WHILE loop, both execute while the condition is true.

Advantages of Pseudocode:

1. Writing of pseudocode involves much less time and effort.
2. Pseudocode reduces complexity.
3. It is easier to modify the pseudocode of program logic whenever program modifications are necessary.
4. Converting pseudocode to a programming language is much easier and simpler as compared to converting a flowchart or decision table.
5. Using pseudocode increased flexibility.

Disadvantages of Pseudocode:

1. In case of pseudocode, a graphic representation of program logic is not available.
2. There are no standard rules to follow in using pseudocode.
3. Different programmers use their own style of writing pseudocode and hence communication problems occur due to lack of standardization.
4. For a beginner, it is more difficult to follow the logic or write the pseudocode, as compared to flowcharting.

5.7 STRUCTURED DESIGN CONCEPTS

- The structured design is a disciplined approach to computer system design.
- Structured design was developed by Ed Yourdon and Larry Constantine.
- This technique deals with the size and complexity of a program by breaking up the program into a hierarchy of modules which result in a computer program. These modules are easier to implement and maintain.
- Structured system design is a step-by-step methodology which produces a software structure.
- The software structure is the specification of the components of the system [modules] and interconnection between these components.

Features of Structured Design:

- The structured design has following five features :

1. It uses a definition of problem to guide definition of solution.
2. It partitions the system into 'black boxes' and organizes a suitable computer implementation for each black box.
3. It uses graphics to make system understandable.
4. It offers set of strategies for developing a design solution.
5. It offers a set of criteria for evaluation of quality into the design solution.

- In short, structured design produces the systems which are easy to understand, reliable, flexible, long lasting, smoothly developed and efficient to operate. It actually produces inexpensive systems.

5.8 STRUCTURE CHART

- A structure chart is a hierarchy diagram. These charts are used to graphically depict a modular design of a program.
- Specially, they show how the program has been partitioned into smaller more manageable modules, the hierarchy and organization of those modules and the communication interface between modules.
- However, structure chart do not show the internal procedures performed by the module or the internal data used by the module.
- The structure charts concentrates on the black box aspect of modules, their external functions, input and outputs rather than their internal procedures and data.
- The structure of a hierarchical system can be specified using a structure chart is shown in Fig. 5.27.

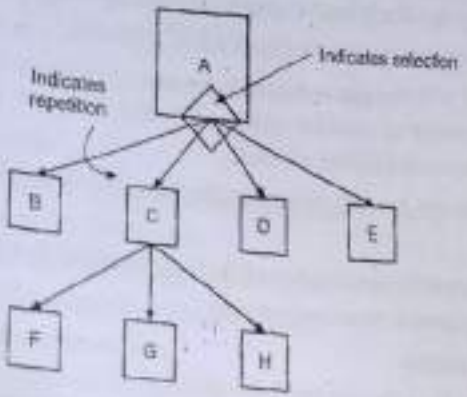


Fig. 5.27: Format of Structure Chart

- To draw the structure chart, the following points are very important:
  1. The structure chart modules are displayed by named rectangles. For example A, B, C etc.
  2. The structure chart modules are presumed to execute in a top to bottom, left-to right sequence.
  3. An arrow from upper module to lower module represents a module call.
  4. An arc shaped arrow located across a line which represents module call means that the module makes iterative call.
  5. A diamond symbol located at the bottom of the module means that the modu calls one and only one of the other lower modules that are connected to th diamond.
  6. Program modules communicate with each other through passing of data.
  7. Program modules may also communicate with each other through passing messages or control parameters called flags.
  8. Library modules are represented on a structure chart as a rectangle containing vertical line on each side.
- The best example of a structure chart is the ATM system available in almost all banks. The different modules which take part into the ATM system are shown following fig.

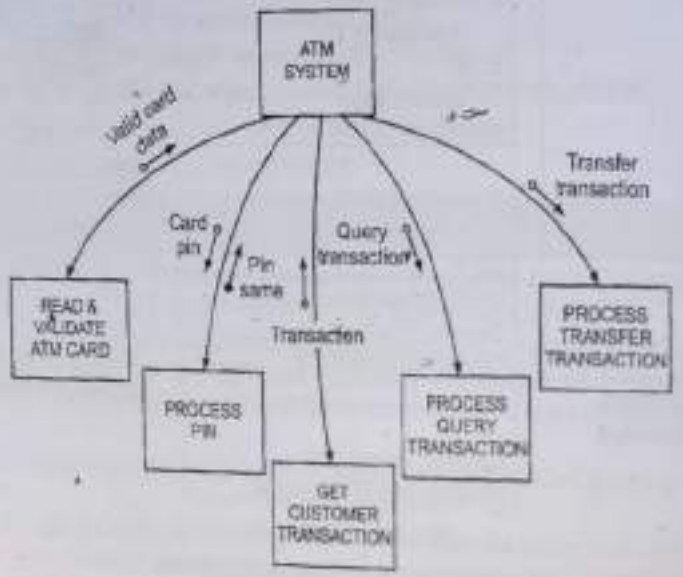


Fig. 5.28: ATM System

Examples:

1. **Phone bill system structure chart:**

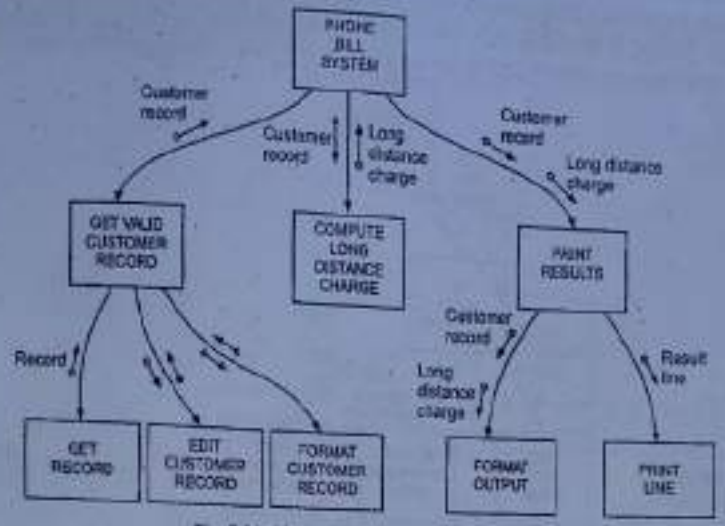


Fig. 5.29. Phone bill system Structure Chart

2. **Payroll system structure chart:**

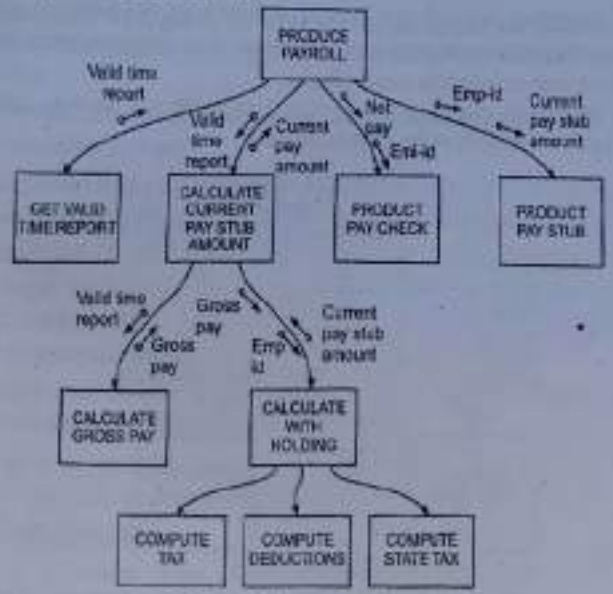


Fig. 5.30: Payroll System Structure Chart

5.7 COUPLING AND COHESION

Module: BBA (CA): W-18

- A module is a logically separable part of a program which is discrete and identifiable with respect to compile and load.
- In the systems using functional abstraction, a module is usually a procedure of function or a collection of these.
- In modular design, some criteria must be used to select module so that it will support well-defined abstraction.

Types of Modules: BBA(CA) : S-19

- The designed shape of a system depends on the types of modules. The module consists of four basic types which determine the direction from that the data flows: Afferent module, Effesent module, Transform module, and Co-ordinate module.
- These four modules tell about 'what you give to a module and 'what you get back from it'.

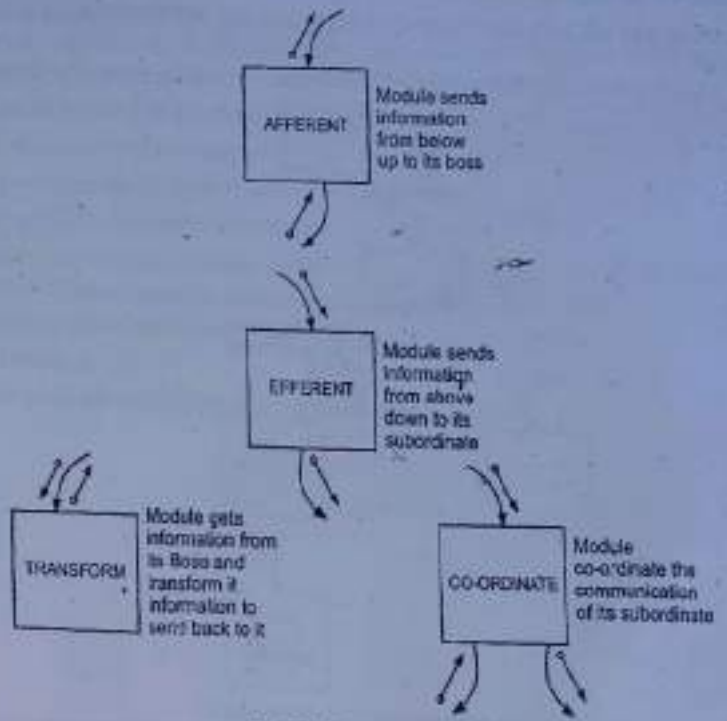


Fig. 5.31: Types of Modules

- The term *afferent* and *efferent* are taken from human anatomy where it is defined as an afferent nerve sends information towards brain and an efferent nerve sending it away from brain. These module types are shown in Fig. 5.31.
  1. **Afferent Module:** This module gets data from subordinate and forwards it to superordinate (boss) modules. These are also called as Input Modules.
  2. **Efferent Module:** This module gets data from supercordinate and forwards it to sub-ordinates. These are also called as Output Modules.
  3. **Co-ordinate Module:** This module manages the flow of data between different subordinate. They are used for selection purpose and in decision making.
  4. **Transform Module:** This module gets data from superordinate. Process that data and again forward it to superordinate modules. These modules are used for processing purpose.
- Let us see the examples which show these module types in detail.

Example 1:

- The module GET VALID CITY NAME, VALIDATE ALPHABETIC FIELD, PRINT ERROR MESSAGE shows efferent module.
- The module GET RECORD, GET VALID CITY NAME and GET CITY NAME shows afferent module.

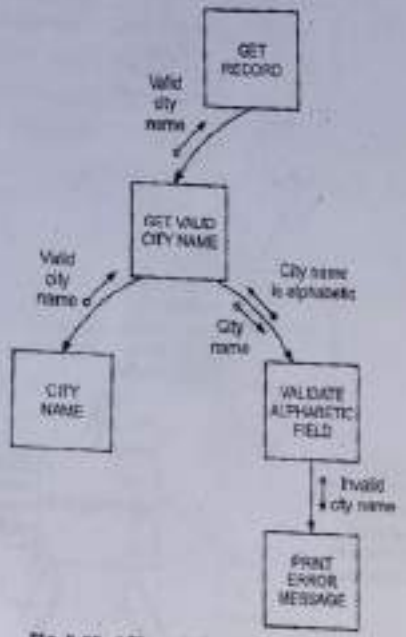


Fig. 5.32: Afferent and Efferent Modules

- This system is not balanced because it prints error message only when it validates alphabetic field. If it takes the name of the city from Module CITY NAME then it does not print error.
- So we can have other alternative design of a simple system. This design is shown in Fig. 5.33. Where, it specifies that when GET VALID CITY NAME executed it should accept City name, validate it. If it is not alphabetic, print error message.

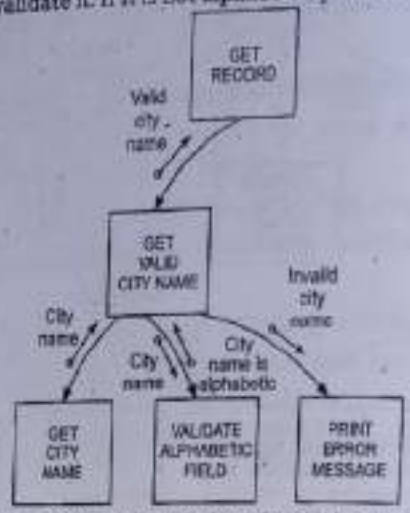


Fig. 5.33: Alternative design of a Simple System

Example 2:

- The example to show the transform module is shown in Fig. 5.34. This module does the pay calculations. GET EMPLOYEE PAY module gets the HOURLY EMPLOYEE TIME REPORT which is send to calculate employee pay.
- This returns the hourly employee pay to GET EMPLOYEE PAY module. This same information again sends back to the control module.

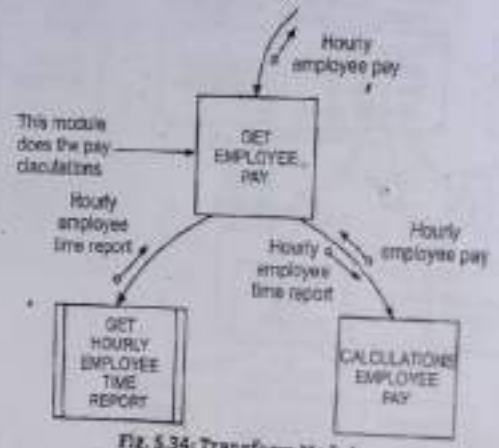


Fig. 5.34: Transform Module

Example 3:

- Fig. 5.35 shows the example of co-ordinate module where the module co-ordinates the communication of its subordinates.

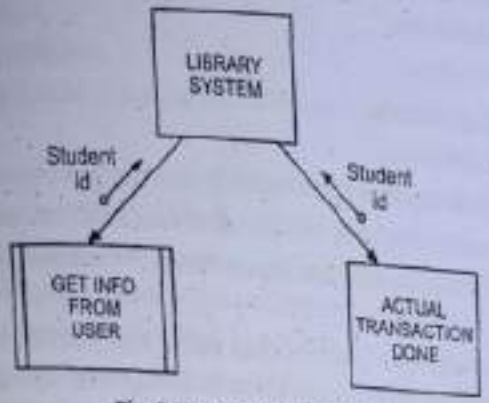


Fig. 5.35: Co-ordinate Module

5.9.1 Coupling

(BBA (CA): S-19)

- In the system which uses functional abstraction, coupling and cohesion are the two criteria's for modularization.
- The two modules are independent if they can function completely without the presence of other modules. If they are independent, they are solvable and modifiable separately.
- In the system it is not possible to have all the modules independent. At least some interaction must be there.
- Coupling refers to the interdependencies between modules. In other words, Coupling between modules is the strength of interconnections between modules.
- Suppose there are two modules A and B. If we know more about module A to understand module B the more closely connected A is to B. The strong interconnection shows highly coupled module and a weak interconnection shows loosely coupled module.
- Coupling is decided at the time of system design because the modules of the software system are created during system design.
- Coupling is an abstract concept and is not easily quantifiable. That is why no formulas can be given to determine the coupling between two modules.

Factors affecting coupling:

- The most important thing between modules is connection between them, the Complexity of Interface and Type of Information Flow.

1. **Complexity of Interface:** Coupling increases with the complexity and insignificance of the interface between modules. For having low coupling we should minimize the number of interfaces per module and also complexity of each interface.
   - If one module passes the information to other module by parameter, it reduces coupling. The degree of coupling is higher if the interface is complex.
2. **Type of information flow along the interfaces:** There are two kinds of information that can flow along an interface. These are Data and Control.
   - Passing or receiving control information means the action of module will depends on the control information. This information makes it more difficult to understand the module and provide its abstraction.
   - Transfer of data information means that the module passes data to another module and also returns the data as output.
   - The coupling is highest if the data is hybrid. It means some data items and some control items are passed between modules.

Ways to minimize Coupling:

- Our main objective is to minimize the coupling i.e. our aim is to make modules independent. Low coupling between modules indicates a well-partitioned system.
- This can be achieved in one of the following three ways:
  1. By eliminating unnecessary relationships.
  2. By reducing the number of necessary relationships.
  3. By easing the tightness of necessary relationships.
- To achieve low coupling between modules, the module should be more independent so that there is less chance for ripple effect (Ripple effect means a defect in one module appears as a symptom in another module).

Types of Coupling:

- The Coupling has various types. These are shown in Fig. 5.36.

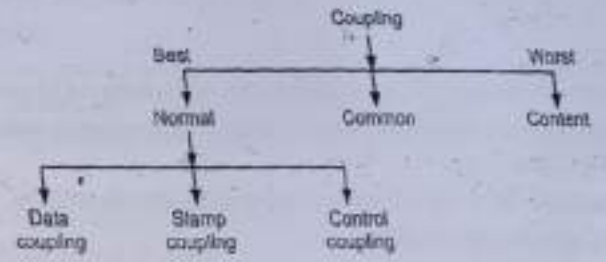


Fig. 5.36: Types of Coupling

1. Normal Coupling

- Consider, there are two modules normally coupled where A calls B and B returns to A. This is shown in Fig. 5.37.

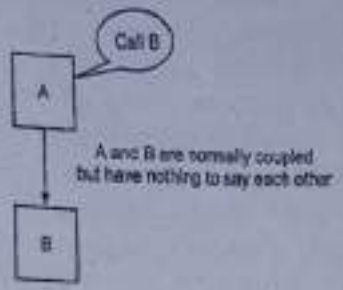


Fig. 5.37: Normal coupling

- All the information passed between them is by means of parameters. It is shown in Fig. 5.38. When no parameters passed, and nothing is written then it makes zero coupling.

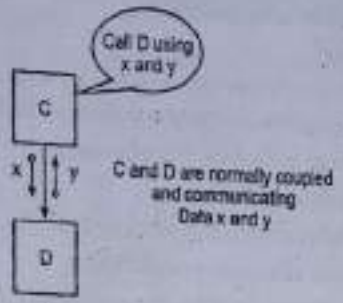


Fig. 5.38: Normal coupling (Data)

- Normal coupling has three main types:
  - (i) Data coupling,
  - (ii) Stamp coupling, and
  - (iii) Control coupling.

(i) Data Coupling

- Fig. 5.39 shows the example of data coupling where data is shared by the module.
- Two modules are data coupled if they communicate by parameters; each parameter is an elementary piece of data.
- This coupling is the necessary communication of data between modules.
- Data coupling is unavoidable and harmless.

For example: The module ACCESS HOUSE AFFORDABILITY uses term, interest rate, sum borrowed to calculate the second module called CALCULATE MORTGAGE REPAYMENT so receives the information about repayment rate.

- Data coupling shows all the best characteristics of coupling.
  - (i) Create narrow connection.
  - (ii) Create direct connection.
  - (iii) Create local connection.
  - (iv) Create obvious connection.
  - (v) Create flexible connection.
- The narrow connection specifies the breadth of an interface between two modules. **For example:** Communicating only two pieces of data from one to another.
- The Direct connections show the interface without referring several other pieces of information first.
- For example, if a module uses piece of data called cust_details, which is defined as name, account number, address and balance. Then a person trying to understand the module connected to the outside world will consider only name, account number and balance. Address may be kept aside.
- The local connection shows the interface with parameter list.
- The obvious connection is easier to understand.

For example: An assembly language routine (A) that communicates with routine B by modifying the contents of a table in B.

- The flexible connection, the information can be changed while passing.

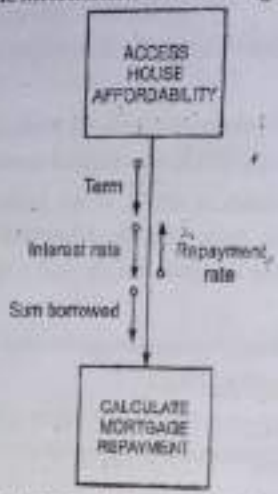


Fig. 5.39: Two data coupled modules

BASIC RENTAL CHANGE NEEDS last three fields but it receives the whole customer rental record.

- Therefore such injudicious stamp coupling broadens interfaces, introduces extra obscurity and reduces flexibility.

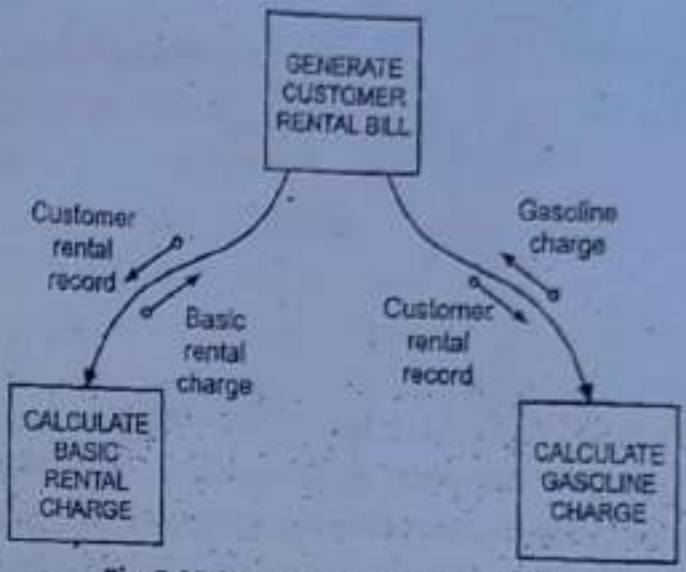


Fig. 5.41: Unnecessary stamp-coupling

(ii) Consider the fragment of a structure chart shown in Fig. 5.42. There are four fields passed to a module. These 4 fields can be replaced by a single data structure shown in Fig. 5.43. This idea of collecting unrelated data items into an artificial data structure is 27 as bundling.

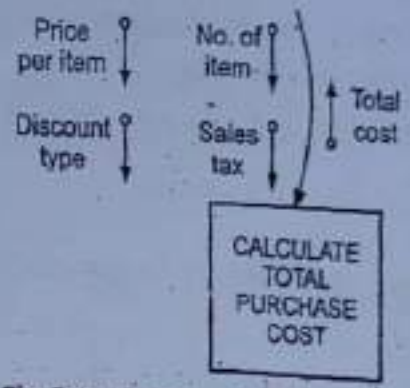


Fig. 5.42: Fragment of structure chart

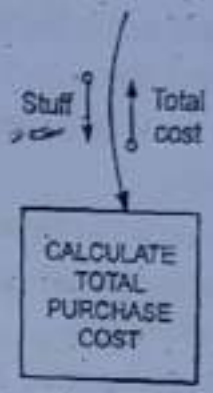


Fig. 5.43: Single data structure

(iii) Control Coupling

- Two modules are control coupled if one passes a piece of information intended to control the internal logic of the other to the other module.
- The value of flag indicates to SYSTEM INPUT/OUTPUT CONTROL which record to read. For example,

  Value of flag is 1 means get next control,

  Value of flag is 2 means master record,

  Value of flag is 3 means Trans record etc.

Fig. 5.44: Two control-coupled modules

- All this leads to some indirectness and obscurity. If a control flag passes downward from a Control to subordinate module, the control module must know something of the internals of the subordinates.

2. **Common (Alias Global) Coupling**

- This is the coupling which stays outside the bounds of good modularity. The two modules are common coupled if they refer to the same global data area.
- FIND PART NAME and REMOVE STOCK are common coupled because they both refer to the same global area, which contains part table and error flag.
- These are the circle symbols called hoc notations; they are not typically used in structured design.
- Common coupling is bad for seven reasons.
  1. A defect in any module using a global area may show up in any other module using that global area.
  2. Global data is referred by explicit name.
     For example: A telephone number editing module receive telephone number from a global piece of data called Tel no, it cannot edit a telephone no which has some other name.
  3. Common coupling introduces and odd kind of remoteness into a system: remoteness in time.
  4. The global area may sometimes be drastically abused as for instance when different modules use the same area to store quote different pieces of information. In Fig. 5.45, error flags mean insufficient stock or no such part number. Due to this extra obscurity, maintenance efforts become hairy.

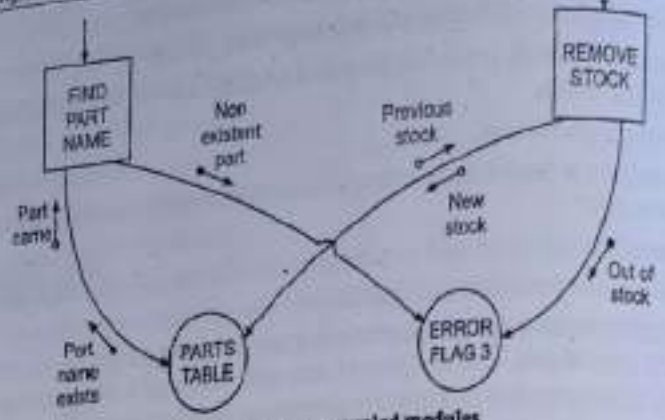


Fig. 5.45: Common-coupled modules

5. Programs with lot of global data are extremely tough for maintenance. Because what data used in a particular module is difficult to know. It is hard to tell what actual coupling is between any pairs of modules. Because one has to know which data is shared?
6. It is difficult to discover what data is changed if the module changed. Even it is also difficult to find out what module changed according to data change.
7. Common coupling is undesirable applies specifically to FORTRAN. In this language, modules refer to the data in common not by name but by location relative to the beginning of the common block.

- The use of small NAMED COMMON areas does largely alleviate this particular problem. However, the use of constant data in a common area in order to reduce redundancy is often permissible. COBOL is not strong in this respect.

3. **Content (Alias Pathological) Coupling**

- Two modules display content coupling if one refers to the inside of the other module anyway.
- For example, if one module branches through another, if one module refers or changes data within another. Such coupling makes non-sense of the concept of black box modules. The higher level languages make it difficult to implement content coupling.
- The problem of content coupling is that it defines the principle of modularity. It brings us back to the undisciplined mess of non-modular coding.

5.9.2 Cohesion

- Cohesion of the module determines how tightly it will be coupled to other modules in a system.
- Cohesion tells up about the bound of internal elements of the module.

- Cohesion gives an idea about whether the different elements of the module belong together in the same module to the designer.
- Cohesion and Coupling are related with each other. More or greater cohesion gives minimum coupling.

Features of Cohesion:

1. Elements that contribute to cohesion are: instructions, groups of instructions, data definition, call of another module
2. We aim for strongly cohesive modules.

- Everything in module should be related to one another - focus on the task.
- Strong cohesion will reduce relations between modules - minimise coupling.
- Cohesion is the measure of the strength of functional relatedness of elements within a module. Elements mean an instruction.
- Designers should create strong, highly cohesive modules whose elements are strongly and genuinely related to one another.
- Cohesion is the second way to tell how well use has partitioned a system into modules.
- A good cohesion is the best way to minimize coupling between modules. The idea of cohesion came to Larry Constantine in mid 1960's.

Types of Cohesion:

- There are seven types of cohesion:
  1. Functional cohesion.
  2. Sequential cohesion.
  3. Communicational cohesion.
  4. Procedural cohesion.
  5. Temporal cohesion.
  6. Logical cohesion.
  7. Coincidental cohesion.
- The functional cohesion is the highest and coincidental cohesion is the lowest level. Functional binding is much stronger them the other levels. The highest level of cohesion is applicable to almost all the modules of the system.

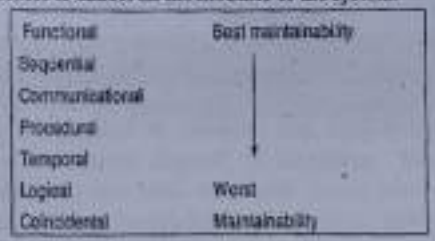


Fig. 5.46: Scale of cohesion

Let's see details of each of type:

1. **Functional Cohesion:**

- It contains elements those contribute to the execution of one and only one problem-related task.

For Examples:

- Compute cosine value of angle.
- Verify alphabetic symbol.
- Read transaction record.
- Compute point of impact of missile.
- Assign seat to airlines passenger.

- Each of these modules has a strong, single-minded purpose. When the control module calls it, carries out just one job to completion without getting involved into any unrelated activity. The systems in which this cohesion occurs are easy to maintain.

2. **Sequential Cohesion**

- This module is one whose elements are involved in activities such that the output data from one activity serves as input data to the next activity.
- The sequence of steps might be something like this:
  1. Take a pot.
  2. Put sugar, tea powder, and water.
  3. Put milk.
  4. Switch on gas.
  5. Put the pot on the flame.
  6. Switch off gas if it boils.
- This group of activities cannot be summed up as a single function. It means the module is not functionally cohesive.
- The sequentially cohesive module has good coupling and easily maintained. The disadvantage is it is not so readily reusable in the same system.

3. **Communicational Cohesion**

- This is the module in which the elements contribute to activities that use the same input or output data. Suppose we want to find facts about a book, we may see following:
  - Find title of book.
  - Find price of book.
  - Find publisher of book.
  - Find author of book.

- All these four activities are related because they all work on the same input data of book which makes the module communicably cohesive. These modules are quite maintainable.
- For example in Fig. 5.47, it is possible that another module in the system wants to find customer name but not interested in his loan balance. This leads to dirty and redundant coupling.
- Another problem with this type of cohesion is to share code among the activities within it. It can make it tough to change one activity without destroying another.

Fig. 5.47: Communication in Cohesion

For example: In Fig. 5.48, EMPLOYEE SALARY REPORT generates reports on all of the employee's salaries and calculates their average salary.

- The modules with communicational and sequential cohesion are similar because they both contain activities organized around the data in the original problem. They have clean coupling because some of their elements are related to the elements in other modules.
- The sequentially cohesive module operates like assembly line i.e. its individual activities must carried out in a specific order. But in communicational cohesive module, order of it is unimportant.

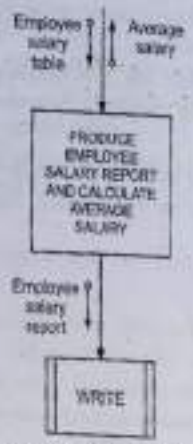


Fig. 5.48: Employee salary report

4. Procedural Cohesion

- Commonly found this module at the top of hierarchy, such as the main program module. This module is one whose elements are involved in different and possibly unrelated activities in which control flows from each activity to the next.

For example:

1. Make a phone call.
2. Take shower
3. Chop vegetables.
4. Set table.

- All above are the modules. They all are happen in a particular way.
- The procedural cohesion modules tend to be composed of pieces of functions which have little relationship to one another. Therefore, this module contains different and unrelated activities and control flows sequentially from one activity to next activity.

1. Temporal Cohesion

- This module is one whose elements are involved in activities that are related in time. Commonly found in initialization and termination modules.
- Elements are basically unrelated, so the module will be difficult to reuse.

For example:

- module initialize
- set counter to 1
- open employee file
- clear error message variable
- initialize array

- The procedural and temporal modules are quite similar with respect to coupling. The difference between them is similar to the difference of sequential and communicational cohesion. This module tends to contain more straight-line code.

2. Logical Cohesion

- This is one whose elements contribute to activities of same general category. In these activities, only selected activities are from outside the module.
- For example, a journey might compile the following list:
  1. Go by car.
  2. Go by train.
  3. Go by boat.
  4. Go by plane.
- They all are means of transport. This module contains a number of activities of the same type. These modules are forced to share only one interface.

5. Coincidental Cohesion

- This module is one whose elements contribute to the activities with no meaningful relationship to one another.

For example:

1. Fix car.
2. Bake cake.
3. Walk dog.
4. Have a juice.
5. Get out of bed.

- It is similar to logical cohesion. These activities are not in data flow and not by flow of control. This is rare. It causes misguided attempts to save time or memory. They have no well-defined functions.

Table 5.3: Differentiate between Coupling and Cohesion

Coupling	Cohesion
1. It is an inter-Module concept.	1. It is an intra-module concept.
2. Coupling shows the relationships between modules.	2. Cohesion shows the relationship within the module.
3. It shows the relative independence among modules.	3. It shows the module's relative functional strength.
4. While creating, you should aim for low coupling, i.e., dependency among modules should be less.	4. While creating you should aim for high cohesion, i.e., a cohesive component/ module focuses on a single function (i.e., single-mindedness) with little interaction with other modules of the system.
5. In coupling, modules are linked to the other modules.	5. In cohesion, the module focuses on a single thing.

5.10 CASE STUDIES (BBA(CA) & BCA: S-18, 19 W-18)

1. Company ABC sells merchandise to wholesale and retail outlets. Wholesale customers receive a two percent discount on all orders. The company also encourages both wholesale and retail customers to pay cash on delivery by offering a two percent discount for this method of payment. Another two percent discount is given on orders of 50 or more units. Each column represents a certain type of order. Draw decision table.

Solution: Decision table for above cases, is given below:

Less than 50 units ordered	Y	Y	Y	Y	N	N	N	N
Cash on Delivery	Y	Y	N	N	Y	Y	N	N
Wholesale Outlet	Y	N	Y	N	Y	N	Y	N
Discount Rate 0%				X				
2%		X	X					X
4%	X					X	X	
6%					X			

2. At Christmas, a company pays a gift of money to some of its employees. To be eligible for the gift, an employee must have worked for the company for at least six months. Managers get $500 and other employees get $300 for their first Christmas with the company and $500 thereafter. Draw decision tree for this situation.

Solution:

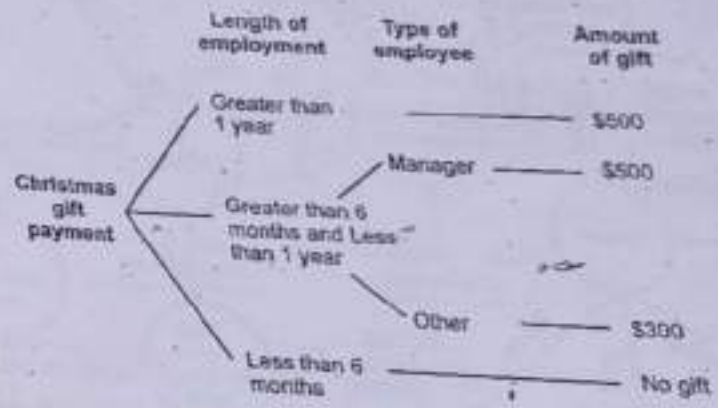


Fig. 5.49

3. Draw DFD for Library Management System.

Solution: Functional components and assumptions:

- **Register user:** New user can register.
- **Book issue:** Here the books will issue to the user.
- **Book return:** Here the books are returned.
- **Search/view book details:** It is used to search and view the books.
- **Update book details:** Here the details of the books will be updated.
- **Payment:** Here payment should be done in manual.

- A context flow diagram is 0^{th} level DFD. It only contains one process node that generalizes the functions of the entire system in relationship to external entities. In context flow diagram the entire system is treated as a single process and all its input, output sinks and source are identified and shown.

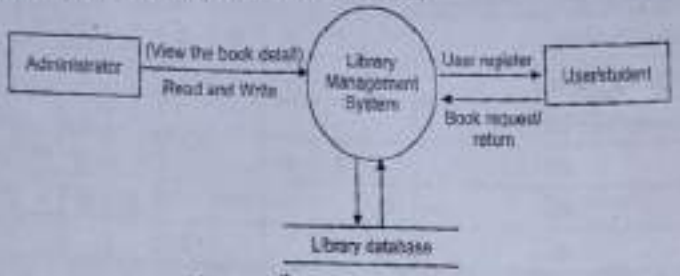


Fig. 5.50: 0^{th} level Data flow Diagram

A data flow diagram is a graph showing the flow of data values from their source in objects through process that transform them to their destination in other objects.

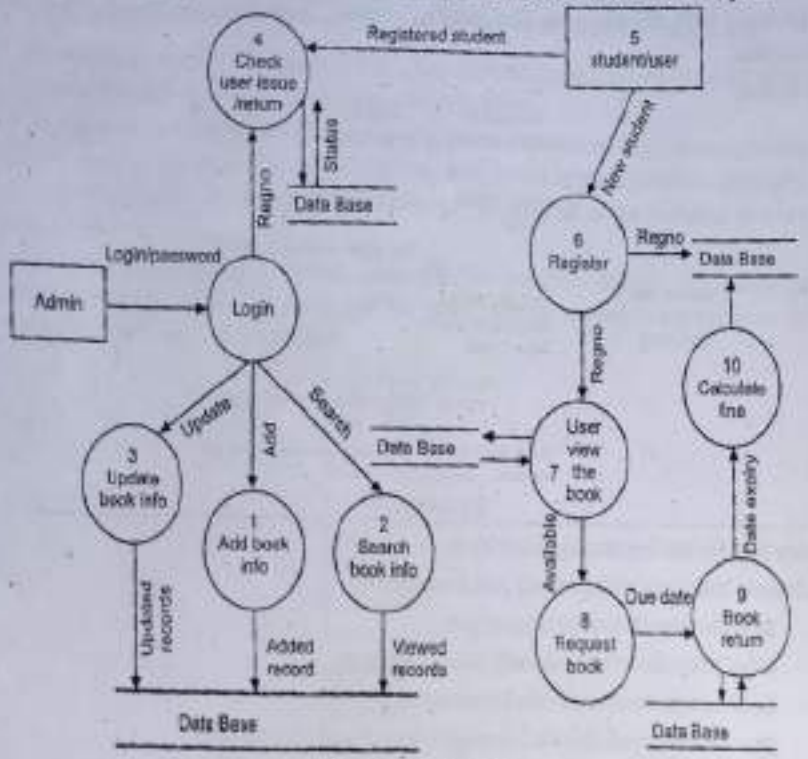


Fig. 5.51: DFD Level 1

Description of Components:

Function Component 1:

- **Registration:** New user can register.
- **Input:** user details.
- **Process definition:** Processing information and stored in the database.
- **Output:** User details updated in the database.

Function Component 2:

- **Book issue:** Here the books will issue to the user.
- **Input:** Book ID.
- **Process definition:** Searching books.
- **Output:** Search and retrieving book information.

Function Component 3:

- **Book return:** Here the books will return.
- **Input:** Book ID.
- **Process definition:** Checking book details.
- **Output:** Book is returned.

Function Component 4:

- **Search/view book details:** It is used to search and view the details of the book.
- **Input:** Book ID.
- **Process definition:** Searching books.
- **Output:** Details of the book will be shown.

Function Component 5:

- **Update book details:** New book entry can be added.
- **Input:** Book ID.
- **Process definition:** Processing the information.
- **Output:** Update in database.

Function Component 6:

- **Payment:** Here payment is done.
- **Input:** User ID.
- **Process definition:** Checking user account details.
- **Output:** Book will be issued.

4. Draw DFD for Hotel Reservation System.

Solution:

DFD Level 0:

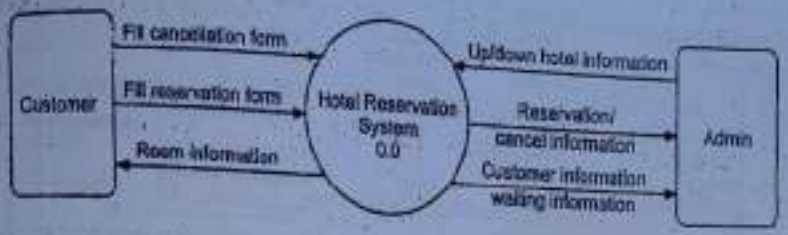


Fig. 5.52

DFD Level 1:

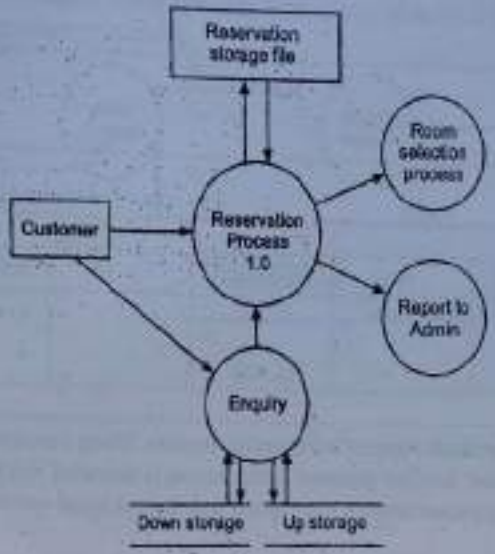


Fig. 5.53

DFD Level 2:

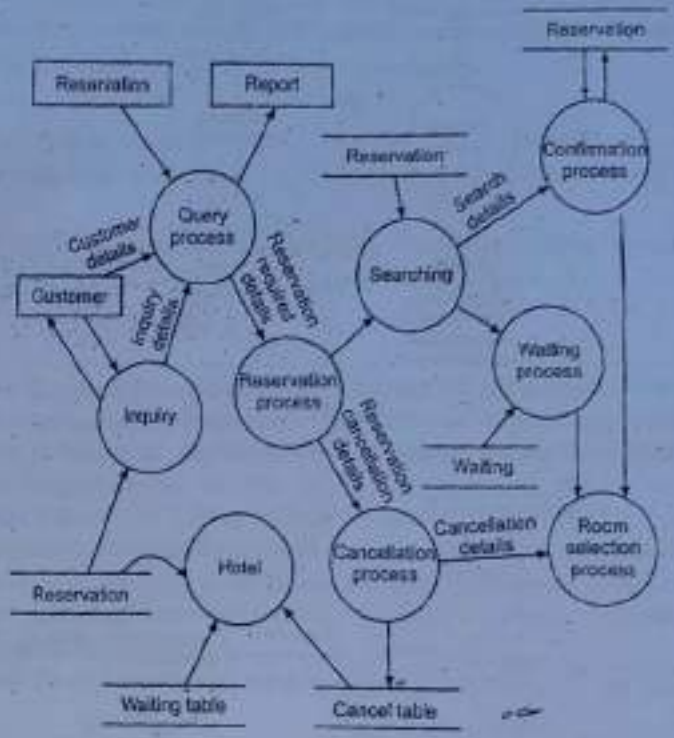


Fig. 5.54

5. Draw DFD for Publisher Ordering System.

Solution: Following are the set of DFDs drawn for the general model of publisher's present ordering system.

DFD Level 0:

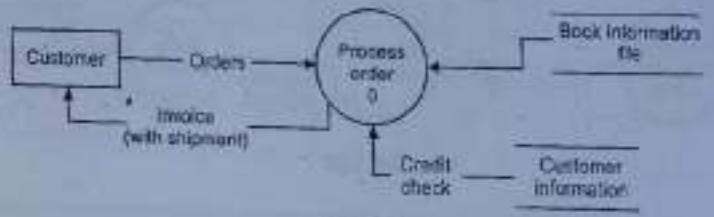


Fig. 5.55

DFD Level 1: Showing publishers present ordering system.

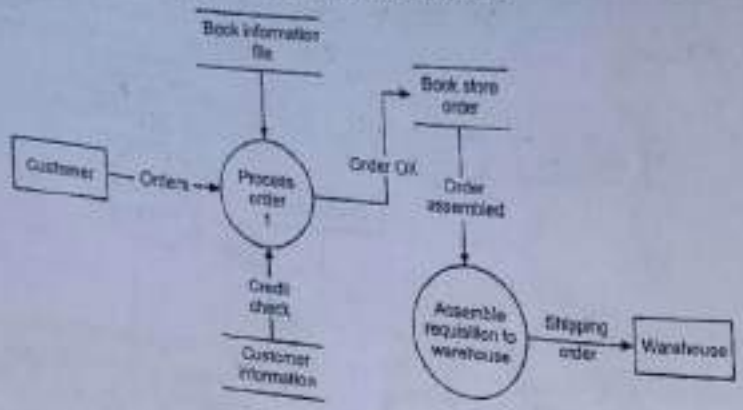


Fig. 5.56

DFD Level 2: Showing order verification and credit check.

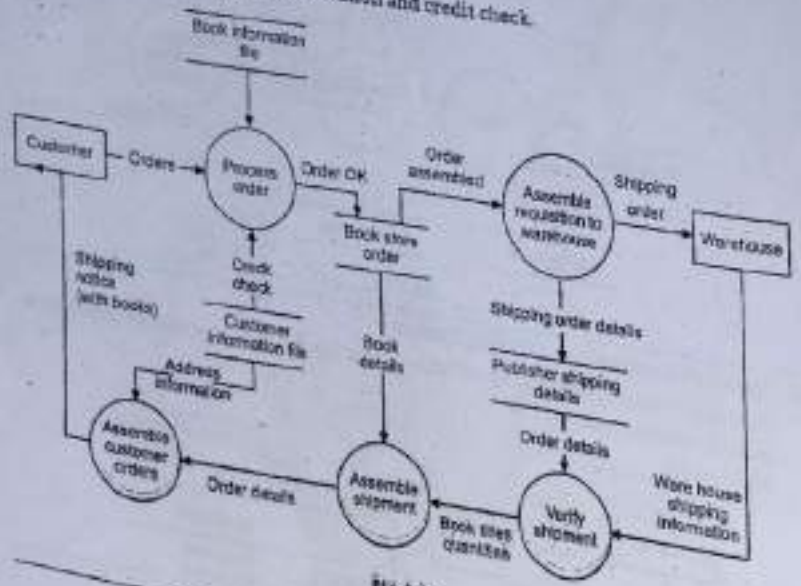


Fig. 5.57

DFD Level 3: Elaborating an order processing and shipping.

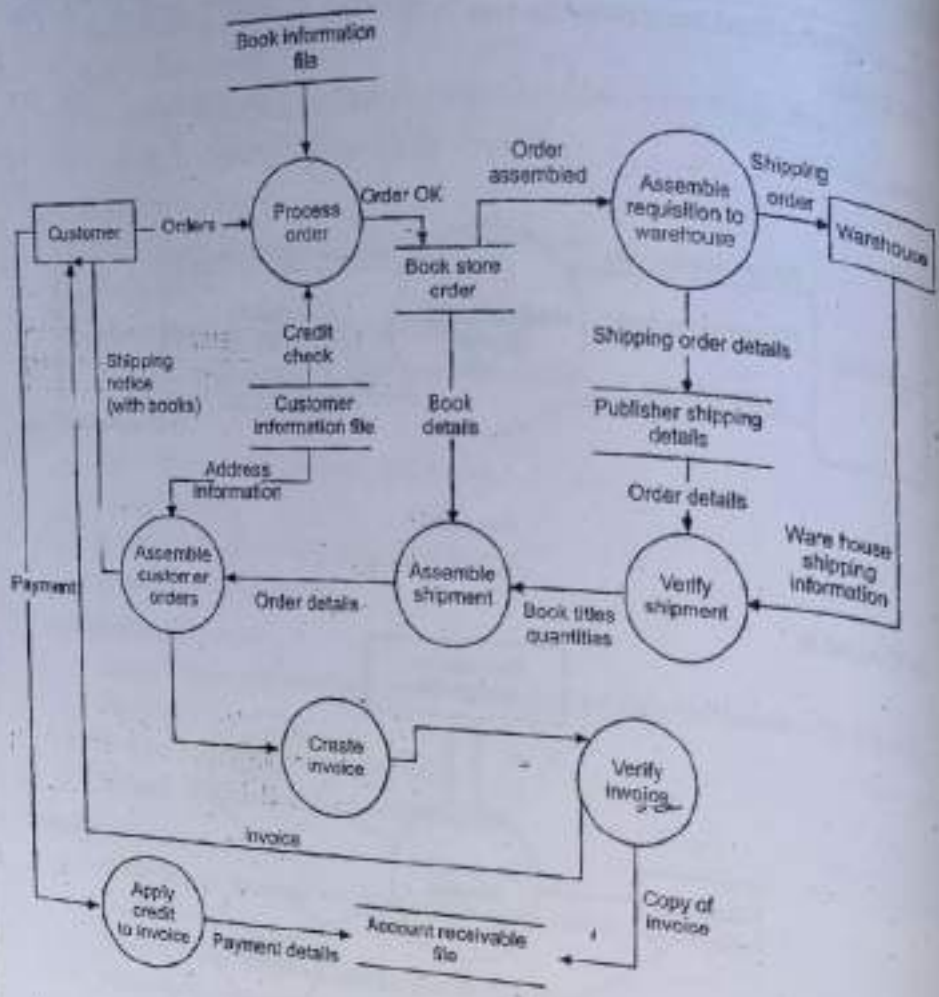


Fig. 5.58

6. **Consider a generalized Student Information System. When a student want to take his/her admission, his/her personal information is recorded and then according to the previously passed exam a class is allotted. Layout input and output design.**

Solution:

Input Design:

SVCP College
STUDENT INFORMATION SYSTEM

Student number:
Student name:
Address:
City:
Sex: ◉ Male ◉ Female
Blood group:
Previous exam. passed:
Percentage:
Date:
Pin code:
Age:
Last collected attended:
Total marks
Out of:
Wanted to take admission in :
First | Previous | Next | Last
Add | Modify | Delete | Save | Cancel | Exit

Fig. 5.59

Output Design: A report which gives the information of student admitted for each class and the total number of students.

SVCP College

Date:

Faculty:

Student number	Name	Admission date	Class	Percentage

Total students:

Grand total:

Authorised signature

Fig. 5.60

7. **Develop a pseudocode to compute wages and to prepare pay salary and absence report. If hours worked are less than 48, prepare Absence Report and compute wages by multiplying hours worked by hourly rate. If hours worked > 48 compute wages as (hours worked × hourly rate) + (hours worked - 48) X 2.0 × hourly rate.**

Solution: Pseudo-code for above question is given below:

```
IF hours worked < 48
   Produce absence report
   Compute wages by multiplying hours worked times hourly rate.
  ELSE
   Compute wages by multiplying hourly rates times 48,
    multiplying 2.0 hourly rate times hours worked over 48, and adding
    the two amounts.
ENDIF
```

8. **Consider the recruitment policy of ABC Software Ltd. If the applicant is a BE then recruit otherwise not. If the person is from Computer Science, put him/her in the software development department and if the person is from non-computer science background put him/her in HR department. If the Person is from Computer Science and having experience equal to or greater than three years, take him/her as Team leader and if the experience is less than that then take the person as Team member. If the person recruited is from non Computer Science background, having experience less than three years, make him/her Management Trainee otherwise Manager.**

Solution: The decision table for the problem stated above can be drawn below:

Condition Stub	Conduction Entry							
	1	2	3	4	5	5	7	8
Is person BE?	Y	N	Y	N	Y	N	Y	N
Is person Comp. Sci. Grand CS?	Y	Y	N	N	Y	Y	N	N
Work Experience > = 3	Y	Y	Y	Y	N	N	N	N
Recruit	X		X		X		X	
Don't Recruit		X		X		X		X
Software Department	X				X			
HR Department			X				X	

Contd...

Team Leader	X							
Team Mem					X			
Mgt. Trainee							X	
Manager			X					

Action Stub — Action Entry

This table can further be refined by combining condition entries 2, 4, 6, and 8. The simplified table is displayed below:

BE?	Y	Y	Y	Y	N
CS?	Y	N	Y	N	-
WEX > = 3	Y	Y	N	N	-
Recruit	Y	Y	Y	Y	
Don't Recruit					X
Software Department	Y		Y		
HR Department		Y		Y	
Team Leader	Y				
Team Mem			Y		
Mgt. Trainee				Y	
Manager		Y			

9. Draw DFD for Hospital Management System.

Solution: DFD Level 0: It only contains one process node that generalizes the functions of the entire system in relationship to external entities.

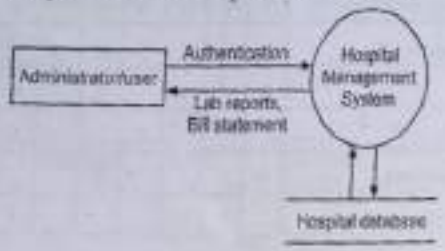


Fig. 5.61

DFD Level 1:

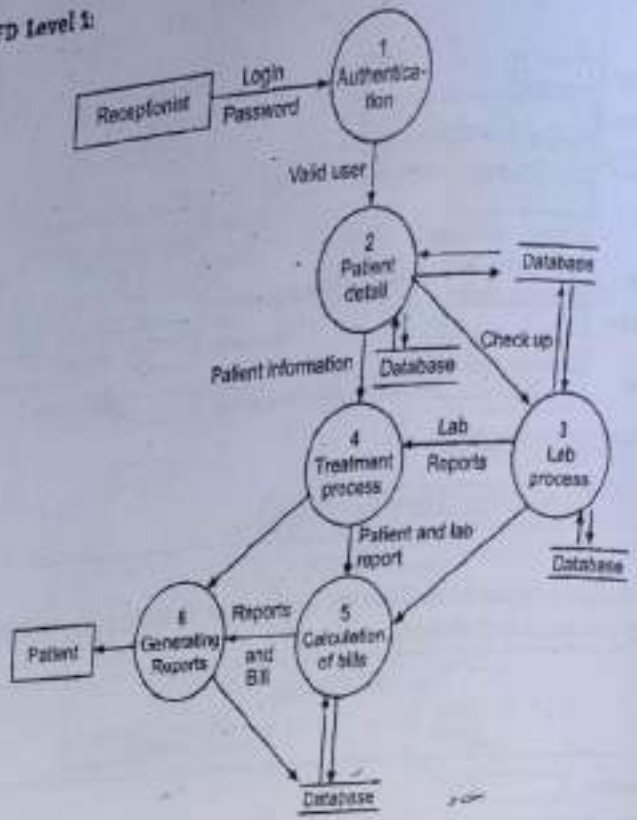


Fig. 5.62

DFD Level 2 and 3:

Fig. 5.63

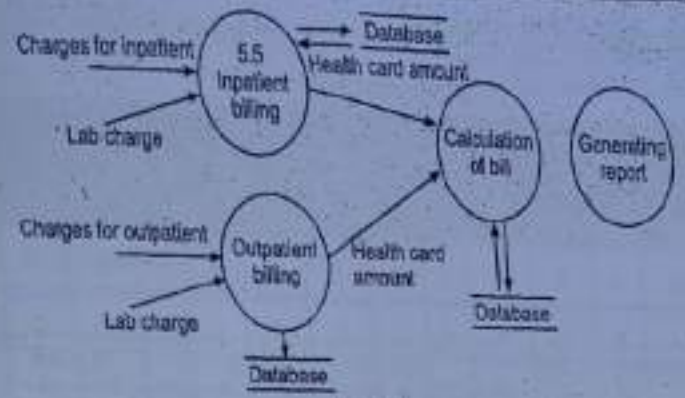


Fig. 5.64

Description of Components:

Administration Module:

Input: User Name, Password

Process Definition: Login to the system

Output: If login is successful then administrator can view and update the records.

Inpatient Module:

Input: Inpatient detail

Process Definition: Information about the treatment given to the patients who are admitted.

Output: Information about the patient.

Interface with other module: The inpatient details are necessary for lab reports and billing.

Outpatient Module:

Input: Outpatient detail

Process Definition: This module has the information about the treatment given to the patient who are came to check up.

Output: Information about the patient.

Interface with other module: The outpatient details are necessary for lab reports and billing.

Lab Module:

Input: Patient ID, Category

Process Definition: Laboratory reports of the patient.

Output: Lab report and charge.

Interface with other module: This module requires information from inpatient and outpatient modules.

Billing Module:

Input: Patient ID

Process Definition: Calculates bill and deduct amount if any card facility is available.

Output: Bill.

Interface with other module: Billing module requires information from inpatient and outpatient modules.

10. Draw DFD for College Management System.

Solution: The DFDs for college management system are given below.

DFD Level 0:

Fig. 5.65

DFD Level 1:

Fig. 5.66

DFD Level 2:

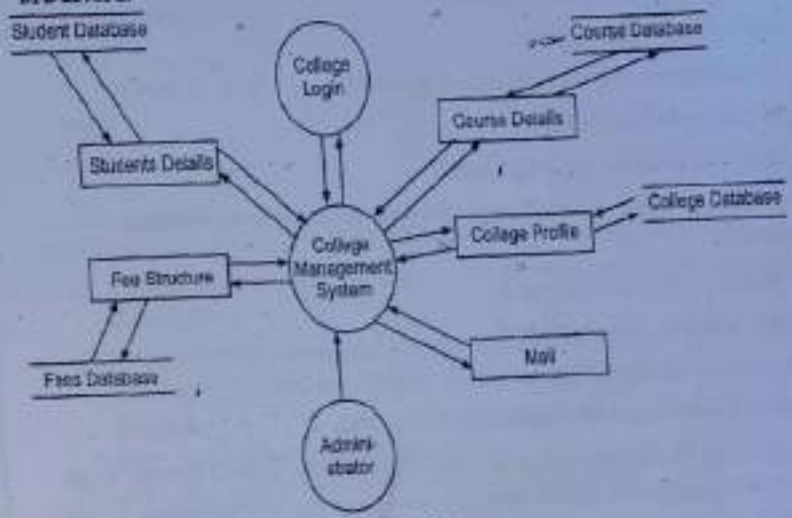


Fig. 5.67

11. A company takes fixed deposits from its employees, share holders and the general public. The FDs may be for the cumulative scheme, or the quarterly interest paid scheme, and for a one year period, or a three year period. For the one year period, the interest rate is 14% p.a. for the quarterly interest scheme, and .5 % higher for the cumulative scheme. For the three year period, the quarterly scheme interest rate is 15.5% and the cumulative scheme is 17%. The incentive is a one time 2% for the general public, and 3% for employees and share holders. Employees investing in FDs of three year period also get an additional incentive of .5%. Give DT for these situations.

Solution:

The conditions are:

- Type of investor : E - Employee
  S - Share holder
  P - General public
- Period of FD : O - One year
  T - Three years
- Scheme : Q - Quarterly interest
  C - Cumulative

Possible actions are given below:

- Interest rate payable-could be 14, 14.5, 15.5 or 17%
- Incentive payable-could be 2% or 3%.
- Additional incentive eligibility-Yes/No

Possible combinations of conditions are the combination of the three possible conditions:

- Condition 1 has 3 possible values.
- Condition 2 has 2 possible values.
- Condition 3 has 2 possible values.

Total combinations possible are therefore $3 \times 2 \times 2 = 12$.

We can make the decision table using extended entries for "interest payable" to get the table shown below. A limited entry table would need the "interest payable" action to be split as four actions (like interest is 14%, interest is 14.5%, interest is 15.5% and interest is 17%).

A Decision Table

Rules / CONDITIONS	1	2	3	4	5	6	7	8	9	10	11	12
Type of investor	E	E	E	E	S	S	S	S	P	P	P	P
Scheme	Q	Q	C	C	Q	Q	C	C	Q	Q	C	C
Period	O	T	O	T	O	T	O	T	O	T	O	T
ACTIONS												
Interest payable	14	15.5	14.5	17	14	15.5	14.5	17	14	15.5	14.5	17
Incentive payable	3	3	3	3	3	3	3	3	2	2	2	2
Additional incentive eligible	N	Y	N	Y	N	N	N	N	N	N	N	N

12. Consider a Hospital Information System. When a patient admits, his/her personal information is recorded. Layout input and output design.

Solution:

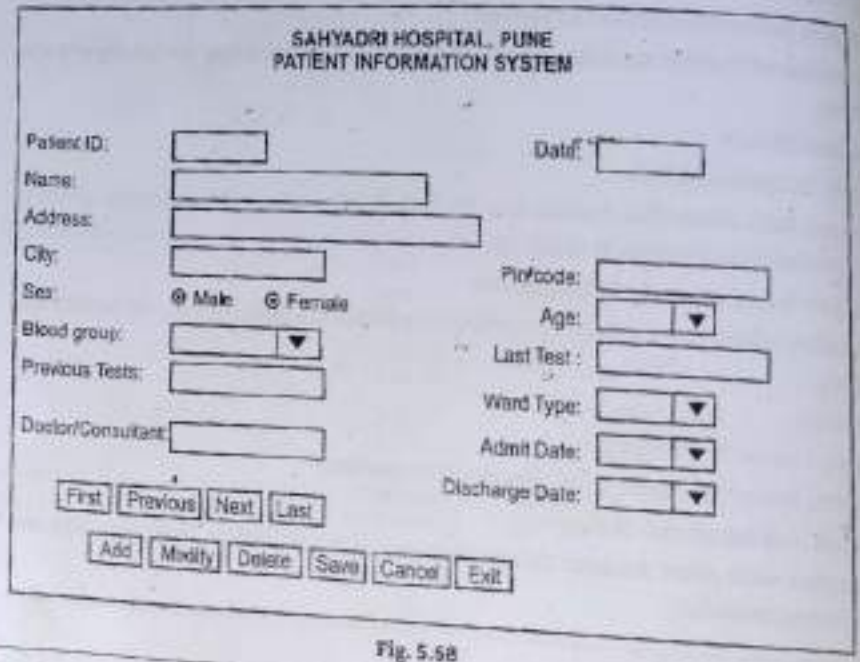


Fig. 5.58

SAHYADRI HOSPITAL
PUNE

Date:

Doctor/Consultant:

Patient ID	Name	Admit date	Tests	Discharge Date

Total Days:

Bill:

Remark

Fig. 5.69

13. Draw Decision tree for Loan Granting Policy.

Solution:

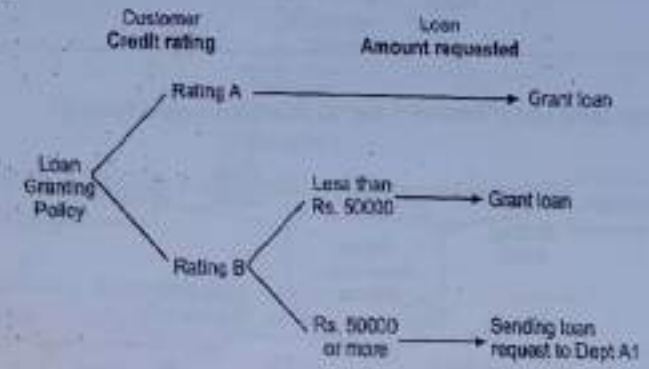


Fig. 5.70

14. Write Pseudocode to find given number is Even or Odd.

```
BEGIN
DEFINE Integer num
    DISPLAY "Enter a number:"
```



```
READ num
IF num%2=0
    DISPLAY "'num' is an even number"
ELSE
    "'num' is an odd number"
ENDIF
END
```

15. Write pseudocode to find the largest of three numbers.

```
BEGIN
READ a, b and c
IF a is greater than b THEN
IF a is greater than c Then
    DISPLAY a
ELSE
    DISPLAY c
ENDIF
ELSE
IF b is greater than c Then
    DISPLAY b
Else
    DISPLAY c
ENDIF
ENDIF
END
```

16. A Co-operative bank XYZ will grant loans under the following conditions :

(1) If a customer has an account with the bank and has no loan outstanding, loan will be granted.

(2) If a customer has an account with the bank but some amount outstanding from previous loans, then loan will be granted if special management approval is obtained.

(3) Reject loan applications in all other cases.

Draw decision tree and decision table for the above case study.

Solution:

Decision Tree:

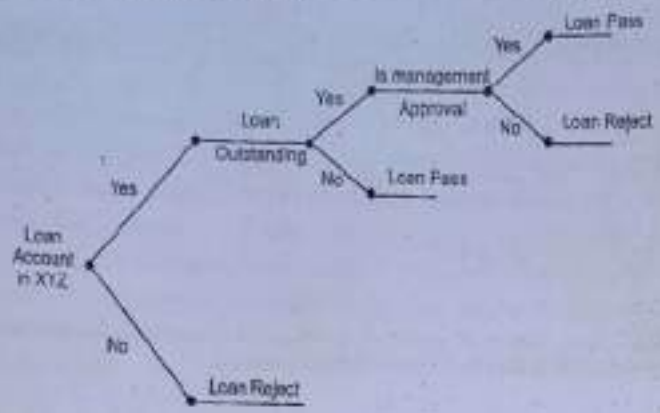


Fig. 5.71

Decision Table:

CONDITIONS	Rule 1	Rule 2	Rule 3	Rule 4
Loan Account in XYZ	Y	Y	Y	N
Outstanding Loan	Y	N	Y	N
Management Approval	Y	N	N	N
ACTIONS				
Loan Pass	X	X		
Loan Reject			X	X

17. Consider the following Case Study:

An insurance company uses the following rule to determine the eligibility of a driver for insurance. The driver will be insured if :

(i) The driver lives in a city with population < 10,000 and he is married.

(ii) The driver lives in a city with population < 10,000 and he is married and his age is above 30 years.

(iii) The driver lives in a city with population is 10,000 or more it is married female.

(iv) The driver is male over 30 years.

(v) The driver is married and under 30.

Draw decision tree for above case study.

Solution:

Decision Tree:

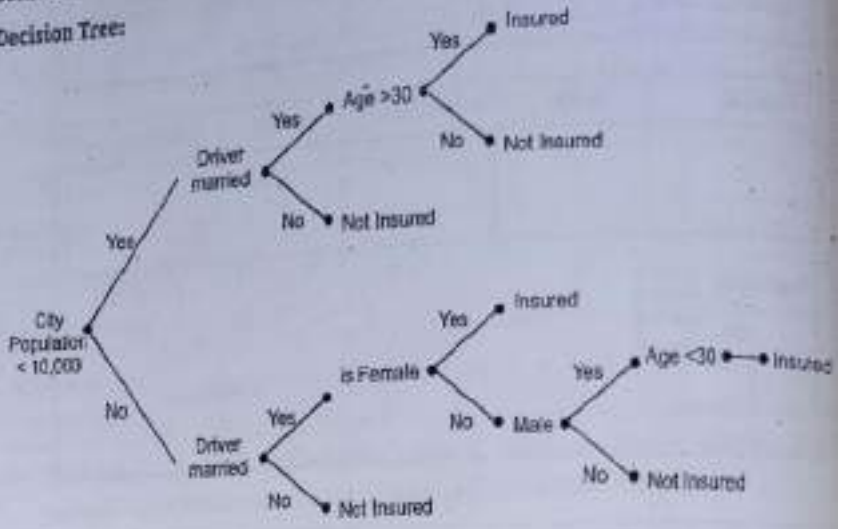


Fig. 5.72

18. Draw Context Level and 1st Level DFD for "Airline Reservation System".

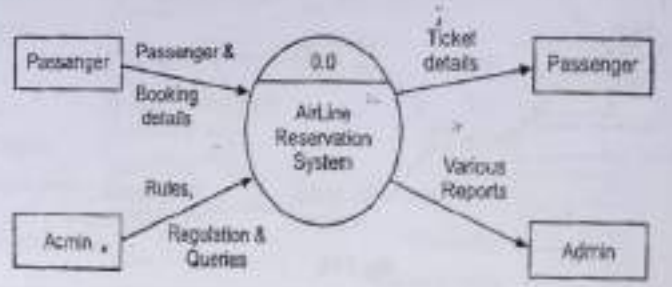


Fig. 5.73

First Level DFD:

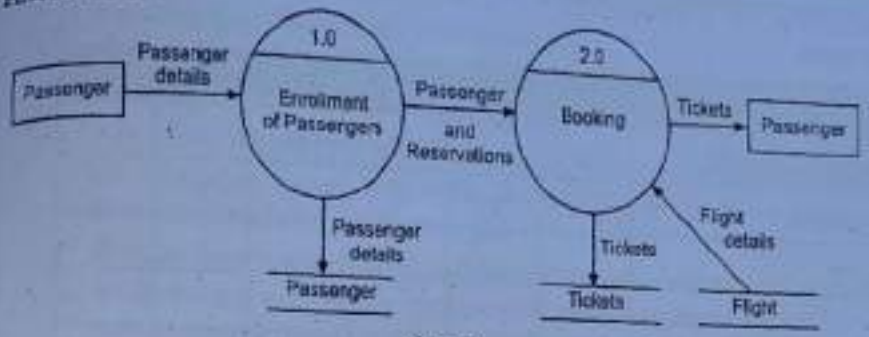


Fig. 5.74

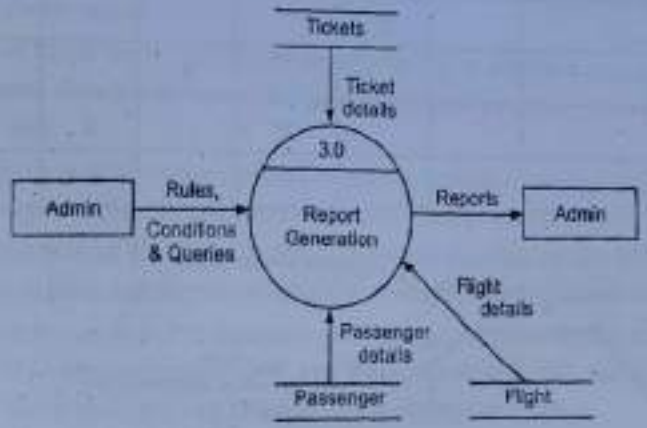


Fig. 5.75

19. Data Flow Diagram for Online Examination for Internal Evaluation.

(i) Context Level DFD:

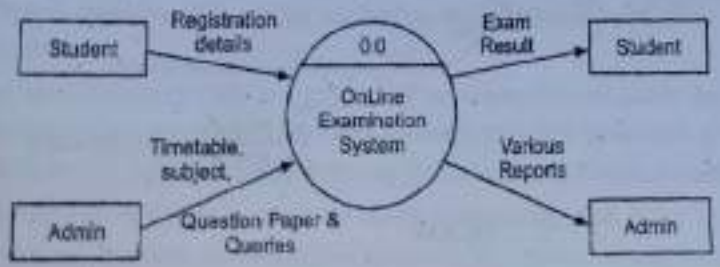


Fig. 5.76

(ii) First Level DFD:

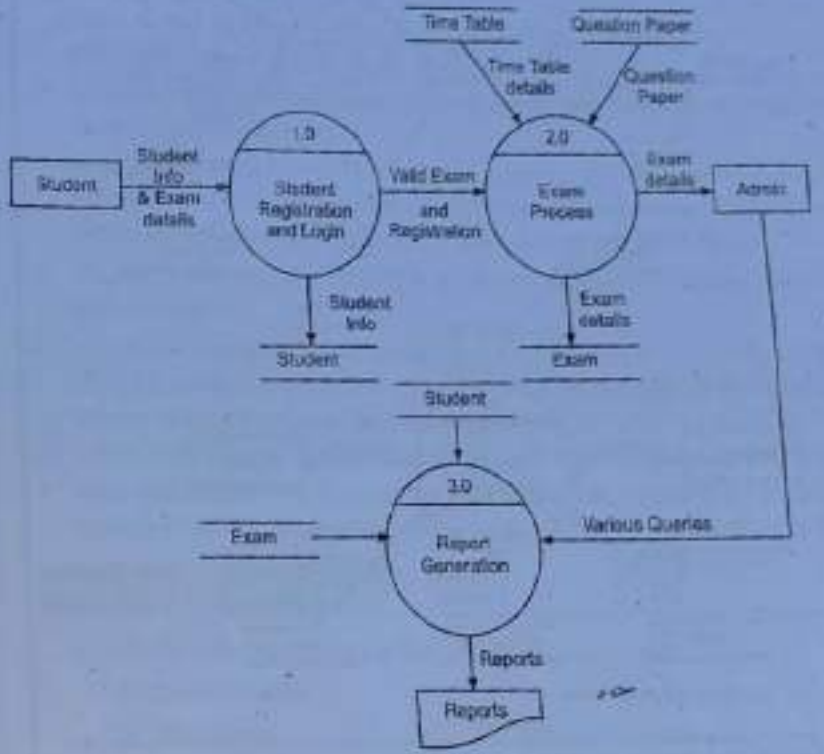


Fig. 5.77

20. Draw Context Level and 1st Level DFD for "Hostel Management System".

(i) Context Level DFD:

Fig. 5.78

(ii) First Level DFD:

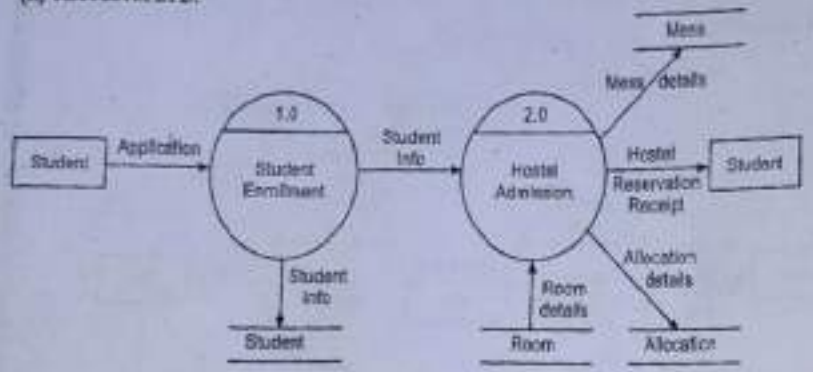


Fig. 5.79

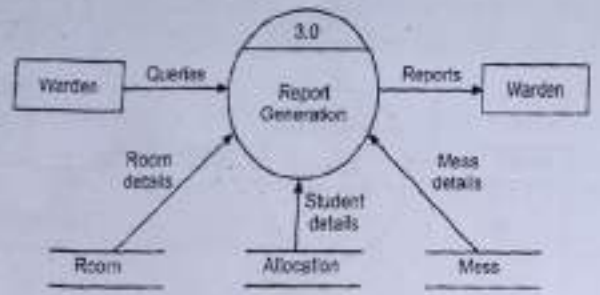


Fig. 5.80

21. Draw Decision Tree and Decision Table for the following Case Study.

An organization decides to give Diwali Bonus to all the employees. For this the management has divided the employees into three categories namely Administrative Staff (AS), Office Staff (OS), Workers (W) and considered the rules given below.

(i) If employee is permanent and in the 'AS' category the bonus amount is three months salary.

(ii) If employee is permanent and in 'OS' category, bonus amount is two months salary.

(iii) If employee is permanent and in 'W' category, the bonus amount is one months salary.

(iv) If employee is temporary then half of the amount is given to them as per the permanent employee's bonus amount.

Decision Table:

Employee Category	Permanent			Non Permanent		
Administrative Staff(AS)	Y		Y	Y		
Office Staff(OS)		Y			Y	
Workers(W)			Y			Y
Bonus						
3 months salary	X					
2 months salary		X				
1 ½ months salary				X		
1 months salary			X		X	
½ months salary						X

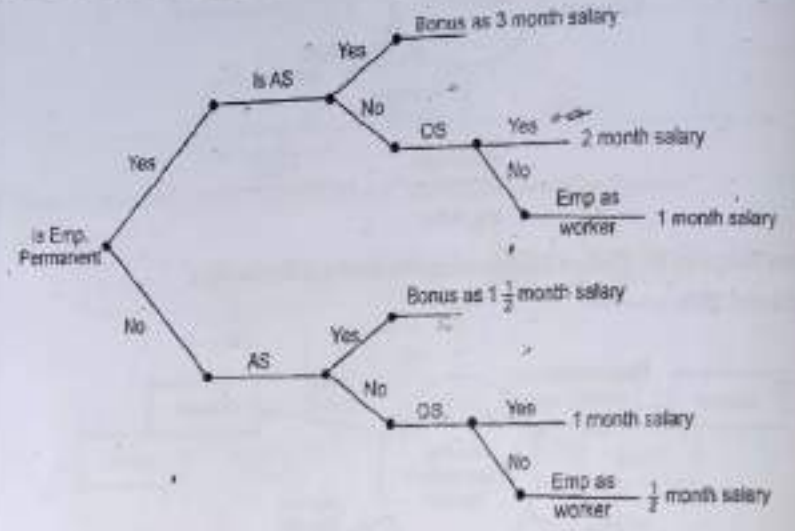


Fig. 5.82

Summary

- Software analysis and design includes all activities, which help the transformation of requirement specification into implementation.
- Analysis and design tools enable a software engineer to create models of the system to be built.
- A Data Flow Diagram (DFD) is a graphical representation of the "flow" of data through an information system. It describes the system's data and how the processes transform the data in a graphical manner.
- DFDs are either Logical or Physical. Logical DFD concentrates on the system process, and flow of data in the system. For example in a Banking software system, how data is moved between different entities. While Physical DFD shows how the data flow is actually implemented in the system. It is more specific and close to the implementation.
- A rectangle in DFD denotes an external entity. It defines a source or destination of system data. It can represent a person, group of people, department, or some other system.
- A circle in DFD denotes a process or activity. It is also known as a bubble. It shows how the system transforms inputs into outputs. Each process is named.
- A line with an arrowhead in DFD denotes the direction of data flow. The input to, or output from, a given process, which is associated with each arrow in a DFD.
- Open Rectangle in DFD denotes a store that is used to model collection of data. It may refer to files or databases, or data stored on optical disk, etc. It is shown by two parallel lines with the name of the data store between them.
- A Decision Table (DT) represents conditions and the respective actions to be taken to address them, in a structured tabular format. DT is a powerful tool to debug and prevent errors.
- Decision tree are graphical representation methods of representing a sequence of logical decisions. It is mainly used when decisions need to be taken or for defining policies.
- A Data Dictionary (DD) is a structured repository of data about data. It is a set of accurate definitions of all DFD data elements and data structures. A data dictionary defines each term encountered during the analysis and design of a new system.
- Structured design is a data-flow based methodology that helps in identifying the input and output of the developing system. It minimizes the complexity and increases

the modularity of a program. It also helps in describing the functional aspects of the system.

- Coupling and cohesion are terms which occur together very frequently. Coupling refers to the interdependencies between modules, while cohesion describes how related the functions within a single module are.
- A structure chart (module chart, hierarchy chart) is a graphic depiction of the decomposition of a problem.
- Inaccurate input data are the most common cause of errors in data processing. Errors entered by data entry operators can be controlled by input design. Input design is the process of converting user-originated inputs to a computer based format.
- The goal of designing input data is to make data entry as easy, logical and free from errors as possible.
- Computer output is the most important and direct source of information to the user. Efficient, intelligible output design should improve the systems relationships with the user and help in decision making. A major form of output is a hard copy from the printer. Printouts should be designed around the output requirements of the user.
- The media devices used for output are MICR, line, matrix and daisy wheel printers, Computer output microfilm, CRT screen, graph plotters and audio response.

Check Your Understanding

1. Which of the following is not the tool for analysis?
   (a) Data dictionary (b) Data flow diagram
   (c) Data science (d) All of the above
2. A decision table made of ____ types of entries.
   (a) 2 (b) 1
   (c) 4 (d) None of these
3. Which are components of data dictionary?
   (a) Data element (b) Data structure
   (c) Data flow (d) All of the above
4. ____ is a tabular method for describing the logic of the decisions to be taken.
   (a) Decision tables (b) Decision free
   (c) Decision method (d) Decision data

5. Pseudocode is also known as _____.
   (a) Program Design Language (b) Program Development Language
   (c) Procedure Development Language (d) Program Design Layout
6. A data store is represented in data flow diagram as _____.
   (a) Rectangle (b) Square
   (c) Open rectangle (d) Open square
7. Which tool is use for structured designing?
   (a) Program flowchart (b) Data-flow diagram
   (c) Structure chart (d) Module

ANSWER KEY

1. (c)	2. (c)	3. (d)	4. (a)	5. (a)
6. (c)	7. (c)			

Practice Questions

Q.I: Answers the Following Questions in short.

1. What is meant by analysis and design tools?
2. What is DFD? Enlist its symbols.
3. What is structured design?
4. What is input design? Explain with example.
5. What is output design? Explain with example.
6. Enlist objectives of input design and output design.
7. What is decision tree and decision table?
8. What is coupling? Explain any two types in detail?

Q.II: Answers the Following Questions.

1. Write short notes on: Levels in DFDs.
2. Describe the advantages and disadvantages of DFD.
3. What is pseudocode? State its advantages and disadvantages.
4. With the help of example describe DD and enlist advantages and disadvantages of DD.
5. What are the elements of DD? Explain with example.
6. Write pseudocode for finding factorial of a number.
7. What are logical and physical DFDs? Compare them.
8. Draw DFD Level 0, 1 and 2 for online Railway Reservation system.

9. Draw DFD for college admission system.
10. Write difference coupling and cohesion.

Q.III: Define the following terms.

1. Data Coupling
2. Decision Tree
3. Coupling
4. Decision Dictionary.
5. Input design
6. Output design
7. Condition Stub
8. Action Stub
9. Cohesion
10. Structured Chart

Previous Exams Questions

BBA(CA)

Summer 2018

1. Design a Screen Layout for creating user account on Internet (with personal details, user-id and password, save, cancel commands etc). [8M]

Ans. Refer to Chapter 5.

2. A Mapro Foods Pvt. Ltd. Company is offering certain discount on the total amount of purchase. If the purchasing amount is more than 5,000 and the customer is making the payment within 5 days then company 5% discount on invoice. If the purchase amount is between 3,000 to 5,000 and the customer is making the payment within 5 days then company offers 3% discount. If the amount is less than 3,000 and customer is making the payment within 5 days then no discount offered and customer has to pay full amount. If customer is not able to pay within 5 days then no discount is given.
   Draw decision table and decision tree for the above case. [8M]

Ans. Refer to Section 5.3.2.

3. Case Study: [16M]
   Consider a Hospital Management System in which the Hospital has InPatient Department (IPD), OutPatient Department (OPD) the system maintains patient records and bills of patient it also manages, information of various wards in the hospital like ICU, General, Private, Semi-private and Delux.
   (a) Identify all entities.
   (b) Draw context level diagram.

(c) First level DFD for the system.

Ans. Refer to 'Case Studies' Section : Case 5.

4. What is module ? **2M**

Ans. Refer to Section 5.9.

5. State any two types of coupling. **2M**

Ans. Refer to Section 5.9.2

6. Define Cohesion. Explain the types of Cohesion. **2M**

Ans. Refer to Section 5.9.2.

7. Write short note: Structured Chart. **4M**

Ans. Refer to Section 5.8.

Winter 2018

1. Define Stamp Coupling **4M**

Ans. Refer to Sections 5.9.2

2. Define module. Explain types of modules. **4M**

Ans. Refer to Sections 5.9

3. (a) Design an I/P Screen Layout form to maintain 'order detail information'. **8M**

(b) Material is issued to the department by considering whether the Material Requisition Note (MRN) is signed or not. It contains valid items or not and it is given within 8 Hours or not. Draw Decision Tree and Decision Table for the above case. **(8 M)**

Ans. Refer to Section 5.3.2.

4. Case Study : **16 M**

Consider a system for swimming tank management. Applicant fill the admission form containing details like address, date of birth, age, father's/gardian's name and also submit two photographs, medical certificate and fees. Then swimming tank management issues I-card to the applicant.

(a) Identify all Entities

(b) Draw Context Level diagrams

(c) First Level DFD for the system.

Ans. Refer to Section 5.3.2.

Summer 2019

1. What is Pseudocode ? **2M**

Ans. Refer to Section 5.6

2. What is Efferent Module ? **2M**

Ans. Refer to Section 5.9

3. Define data dictionary. **2M**

Ans. Refer to Section 5.4

4. What is coupling ? Explain different types of coupling. **4M**

Ans. Refer to Section 5.9

5. Design an input form for "Employees Salary Slip" details of the computer. **6M**

If the volume of sales is greater than ₹ 10,000 and advance collected is 70% or more then commission is 20%. If advance collected is less than 70%, then it is 15%. For sales ₹ 10,000 irrespective of the advance collected commission is 12%. For sales less than ₹ 10,000, commission is 10% or 9% based on whether advance collected is 70% more or less respectively. Draw decision table and decision tree for the above case.

6. Short note: Structured chart **4M**

Ans. Refer to Section 5.8

7. Case Study: Indian Bank provides Fixed deposit schemes through which people can deposit money for a certain period of time. The bank pays interest for this period and returns money when FD period is over interest rate depends upon the period.

The depositor may get loan against deposits. A maximum of 75% of the deposit amount as loan amount. **(8M)**

(i) Draw context level diagrams

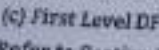
(ii) Draw First level DFD for system.

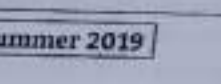
Ans. Refer to Section 'Case Studies'.

BCA (Science)

Summer 2018

1. Identify the symbol used for Data Store in DFD. **1M**

(a)  (b) 

(c)  (d) 

Ans. Refer to section 5.3

2. Define data dictionary (1M)

Ans. Refer to section 5.4

3. State all symbols of DFD. (4M)

Ans. Refer to section 5.3

4. What is Data Dictionary? Explain the importance of Data dictionary (3M)

Ans. Refer to section 5.4

5. Draw context level and 1st level DFD for "Airline Reservation System". (5M)

Ans. Refer to 'case studies' section of chapter 5

6. Explain Decision Table with example. (4M)

Ans. Refer to section 5.2.2

Winter 2018

1. .....is a tabular method for describing the logic of the decisions to be taken. (1M)

(a) Decision Tables
(b) Decision Tree
(c) Decision Method
(d) Decision Data

Ans. Refer to section 5.2.2

2. A context diagram is used ..... (1M)

(a) as the first step in developing a detailed DFD of a system
(b) in system analysis of very complex systems
(c) as an aid to system design
(d) as an aid to system developer and system programmer

Ans. Refer to section 5.3

3. Physical DFD shows "what is going on". State true/false. (1M)

Ans. Refer to section 5.3 (1M)

4. Define data flow diagram.

Ans. Refer to section 5.3 (4M)

5. Draw context level and level '1' DFD for "Hostel Management System". (3M)

Ans. Refer to 'case studies' section of chapter 5

6. Write any two advantages and disadvantages of DFD.

Ans. Refer to section 5.4.2

7. A co-operative bank XYZ will grant loans under the following conditions: (5M)

a. If a customer has an account with the bank and has no loan outstanding, loan will be granted.

b. If a customer has an account with the bank but some amount outstanding from previous loans, then loan will be granted if special management approval is obtained.

c. Reject loan applications in all other cases.

Draw decision tree and decision table for the above case study.

Ans. Refer to 'Case Studies' section of chapter 5

8. Define and explain any four elements of data dictionary. (4M)

Ans. Refer to section 5.4.1

Summer 2019

1. Context level DFD is also called ..... (1M)

(a) Root level (b) Level 0
(c) Level 1 (d) Level 2

Ans. Refer to section 5.3

2. The alternate names used are called ..... (1M)

(a) Aliases (b) Duplicate
(c) Rename (d) None of above

Ans. Refer to section 5.4

3. Define action and task. (1M)

Ans. Refer to section 5.2

4. What is Pseudocode? (1M)

Ans. Refer to section 5.6

5. Enlist the objectives of Input Design. (3M)

Ans. Refer to section 5.5.1

6. Consider the following case study:

(1) An insurance company uses the following rule to determine the eligibility of a driver for insurance. The driver will be insured if:

(i) The driver lives in a city with population < 10,000 and he is married.

(ii) The driver lives in a city with population < 10,000 and he is married and his age is above 30 years.

(iii) The driver lives in a city with population is 10,000 or more it is married female.

(iv) The driver is male over 30 years.

(v) The driver is married and under 30.

(2) Draw decision tree for above case study.

Ans. Refer to 'case studies' section of chapter 5

(3) Draw decision table for above case study.

Ans. Refer to 'Examples' of section 5.2.2.1

7. Draw context level and 1st level DFD for "Hostel Management system". [5M]

Ans. Refer to 'Case Studies' section of chapter 5

8. Explain the types of outputs. [4M]

Ans. Refer to section 5.5.2

9. Explain the term : Data capture. [3M]

Ans. Refer to section 5.5.1

◇◇◇

6... Software Testing

Objectives...

- To understand concepts of software testing.
- To know about various types of testing like System testing, unit testing, integration testing.

6.1 INTRODUCTION

- Testing is the process of evaluating a system or its component(s) with the intent to find whether it satisfies the specified requirements or not.
- Software Testing refers to the process of evaluating software and its components to identify any errors, bugs or errors that might potentially disrupt the functionality of the software.

Need of Software Testing:

- Software testing is very important because of the following reasons:
  1. Testing is necessary in order to provide the facilities to the customers like the delivery of high quality product or software application which requires lower maintenance cost and hence results into more accurate, consistent and reliable results.
  2. Software testing is really required to point out the defects and errors that were made during the development phases.
  3. Testing is essential since it makes sure of the customer's reliability and their satisfaction in the application.
  4. It is very important to ensure the quality of the product or software application.
  5. Testing is required for an effective performance of software application or product.

Testing Objectives and Principles: BBA(CA)-W-25

Software testing has different objectives/principles. The major objectives of software testing are as follows:

1. Finding defects which may get created by the programmer while developing the software.
2. To prevent defects.
3. To make sure that the end result meets the business and user requirements.
4. To ensure that it satisfies the BRS (Business Requirements Specification) and SRS that is (System Requirements Specification).
5. To gain the confidence of the customers by providing them a quality product.

- **Testing software typically involves:**
  1. Executing software with inputs representative of actual operation conditions (or operational profiles).
  2. Comparing produced/expected outputs.
  3. Comparing resulting/expected states.
  4. Measuring execution characteristics (e.g., memory used, time consumed, etc.)

PDCA Cycle:

- Software testing is an important part of software development process. In normal software development there are four important steps, also referred to, in short, as the PDCA (Plan, Do, Check, Act) cycle.
- PDCA cycle is a project planning tool. PDCA cycle is 4-step model for carrying out the change. It is repeated again and again for continuous improvement.
- It is one of the key concepts of quality and it is also called the Deming circle/cycle/wheel.
- PDCA cycle is designed for using as a dynamic model for testing. After completing the turn of one cycle leads to begin the next cycle. Followed by the continuous quality improvement, the process can be reanalyzed and a new test change begins.
- Consider the PDCA (Plan, Do, Check, Act) cycle in Fig. 6.1.

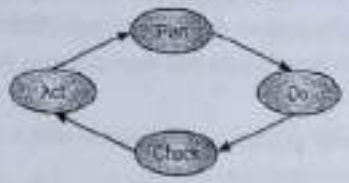


Fig. 6.1: PDCA Cycle

- The basic elements of PDCA cycle in Fig. 6.1 are explained below:
  1. **Plan:** Define the goal and the plan of how to achieve that goal.
  2. **Do/Execute:** Depending on the plan strategy decided during the plan stage we do execution accordingly in this phase.

3. **Check:** Check/Test to make sure that we are moving according to plan and we are getting the desired results.
4. **Act:** During the check cycle if any issues are there then take appropriate action accordingly and revise the plan again.

Testing Terminology:

- **Error:** A computer programmer writes a program. An error occurs in the process of writing a program.
- **Fault and Failure:** A fault is the manifestation of one or more errors. A failure occurs when a faulty piece of code is executed leading to an incorrect state that propagates to the programs output.
- **Bug:** A fault detected in the program while testing time, is also commonly referred to as a bug.
- **Defect:** Defect is a variance between expected results and actual results detected by developer.
- **Test Case:** This is set of inputs, execution conditions, and expected results developed for a particular objective. IEEE defines test case as, "A set of input values, execution preconditions and expected outcomes developed for a particular objects".
- **Test Suite:** This is collection of test cases, typically related by a testing goal or an implementation dependency.
- **Test Driver:** This is class or utility program that applies test cases.
- **Test Harness:** This is system of test drivers and other tools that support test execution.
- **Test Strategy:** This is algorithm or heuristic to create test cases from a representation, implementation, or a test model.
- **Test Plan:** This is a document describing the scope, approach, objectives, resources and schedule of a software testing effort. It identifies the items to be tested, items not to be tested, who will do the testing, which test, approach followed, what will be pass/fail criteria, training needs for team, the testing schedule, etc.
- **Test Script:** This is a testing work product modeling a software program (often written in a procedural scripting language) that executes a test suite of test cases.
- **Test Metrics:** These metrics accomplish in analyzing the current level of maturity in testing and give a projection on how to go about testing activities by allowing us to set goals and predict future trends.
- **Testing:** it is defined as, 'execution of a work product with intent to find a defect'.
- **Test Case Specifications:** This should be developed from the test plan and are the second phase of the test development life cycle. The test specification should explain "how" to implement the test cases described in the test plan.

- **Test Stub:** It is partial temporary implementation of a component (usually required for a component to operate).

6.2 DEFINITION OF SOFTWARE TESTING (BBA(CA):S-18)

- According to ANSI/IEEE 1059 standard, testing can be defined as, "a process of analyzing a software item to detect the differences between existing and required conditions (that is defects/errors/bugs) and to evaluate the features of the software item".

OR

- According to Myers, "testing is technique of software to find the error or mistake in a particular software design".

OR

- As per IEEE, "the process of testing an integrated hardware and software system to verify that the system meets its specified requirements".
- It implies unconditional or conditional, direct or indirect mistakes from our design phase to implementing phase in particular software area.

6.2.1 Life Cycle of Software Testing

- Software testing is a check activity to validate whether the actual results matches with expected results and to ensure that the software system is bug free.
- Software Testing Life Cycle (STLC) is a process of software testing in which specific steps to be executed in a definite sequence to ensure that the quality goals have been met.
- In other words, STLC defines the life cycle or phases of testing process for a software or a system/application. It tells the planned and systematic process of execution to improve and deliver a quality product.
- Testing is a process rather than a single activity. Testing must be planned and it requires discipline to act upon it. The quality and effectiveness of software testing are primarily determined by the quality of the test processes used. The activities of testing can be divided into the following basic steps:
  1. Planning and Control
  2. Analysis and Design
  3. Implementation and Execution
  4. Evaluating exit criteria and Reporting test results
  5. Test Closure activities
- Different companies have different phases in STLC and each activity is carried out in a planned and organized way. These phases of STLC are shown in Fig. 6.2.

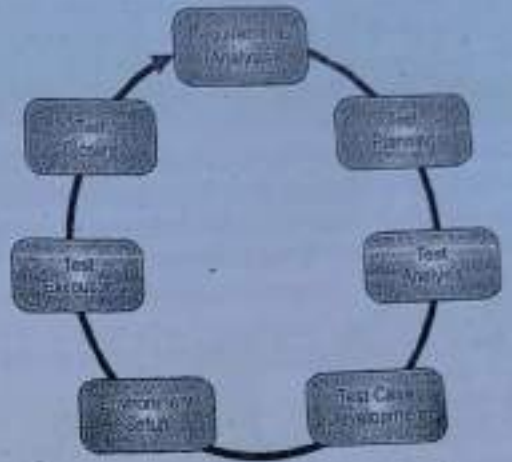


Fig. 6.2: Phases of Software Testing Life Cycle

The phases of STLC are explained below:

- **Phase 1: Requirement Analysis:** In requirement analysis phase, tester analyzes the requirement to find out whether the requirements are testable or not. This phase also identify the scope of the software testing.
- **Phase 2: Test Planning:** In this phase, all the planning related to testing is done and tester identifies the activities and resources that are helpful to meet the testing objectives. A test plan outlines the strategy that will be used to test an application, the resources that will be used, the test environment in which testing will be performed, and the limitations of the testing and the schedule of the testing activities.
- **Phase 3: Test Analysis:** This phase determines the guidelines that have to be tested. It includes identifying the test conditions using the requirements document, any risks involved, and other test criteria.
- **Phase 4: Test Case Development:** This phase involves the actual test case creation. The main objective of this phase is to prepare test cases for an individual unit. These functional and structural test cases cover the functionality, points of verification and validation mentioned in the Test Plan. It also involves specification of test data and automated test scripts creation.
- **Phase 5: Test Environment Setup:** This phase includes the setup or installation process of software and hardware which is required for testing the application. It is a combination of hardware and software environment on which the tests will be executed.

- **Phase 6: Test Execution:** Test execution is the process of executing the code and comparing the expected and actual results. Before starting the Test Execution phase the Test Environment setup should be ready. In Test Execution phase, the test cases are executed in the testing environment. This phase involves manual and automated test case execution.
- **Phase 7: Test Closure:** This phase marks the formal closure of testing. Test Closure is a document that gives a summary of all the tests conducted during the software development life cycle, it also gives a detailed analysis of the bugs removed and errors found. Once testing is completed, matrix, reports, results are documented.

6.2.2 Types of Testing

- There are mainly two broad types of software testing i.e., Manual Testing and Automation Testing.
- Executing the test cases manually without any tool support is known as Manual Testing. Taking tool support and executing the test cases by using automation tool is known as Automation Testing.

1. Manual Testing:

- Manual testing includes testing a software manually, i.e., without using any automated tool or any script.
- In this type, the tester takes over the role of an end-user and tests the software to identify any unexpected behavior or bug.
- There are different stages for manual testing such as unit testing, integration testing, system testing, and user acceptance testing.
- Testers use test plans, test cases, or test scenarios to test a software to ensure the completeness of testing.

Advantages of Manual Testing:

(i) It is cost effective for test run only few times.

(ii) There is no need of programming knowledge.

(iii) The cost of manual/testing is less than automated testing.

(iv) It allows the tester to do more ad-hoc testing.

(v) UI (User Interface) testing can be done accurately with the help of manual testing.

Disadvantages of Manual Testing:

(i) Manual testing have high risk of error and mistakes as it is done manually by testers.

(ii) Scope of manual testing is very limited. Manual testing is not suitable in very large organizations and time bounded projects.

(iii) Comparing large amount of data in case of manual testing is difficult and time consuming.

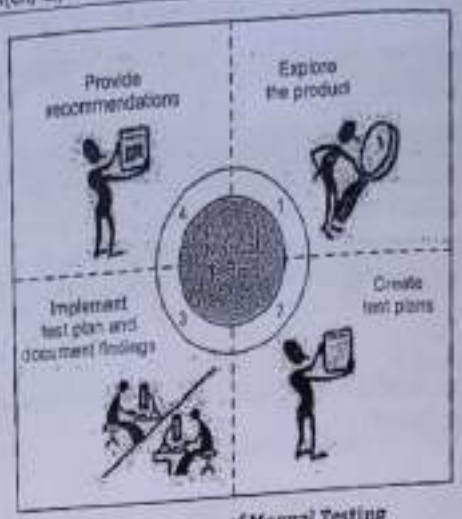


Fig. 6.3: Process of Manual Testing

2. Automation Testing:

- Automation testing, which is also known as 'test automation', is when the tester writes scripts and uses another software to test the product. This process involves automation of a manual process.
- Automation testing is used to re-run the test scenarios that were performed manually, quickly, and repeatedly.

Advantages of Automation Testing:

(i) **Fast:** Runs tests significantly faster than human users.

(ii) **Repeatable:** Testers can test how the website or software reacts after repeated execution of the same operation.

(iii) **Comprehensive:** Testers can build test suites of tests that cover every feature in software application.

(iv) **Reusable:** Tests can be reused on different versions of the software.

(v) **Reliable:** Tests perform precisely the same operation each time they are run thereby eliminating human error.

(vi) **Programmable:** Testers can program sophisticated tests that bring hidden information.

Disadvantages of Automation Testing:

(i) **Expensive Tools:** Buying software automation tools is expensive.

(ii) **Tools still take time:** A considerable amount of time goes into developing the automated tests and letting them run.

(iii) **Set-up Problems:** Automatic tests take more time to set up, which does not allow testing ideas quickly and easily.

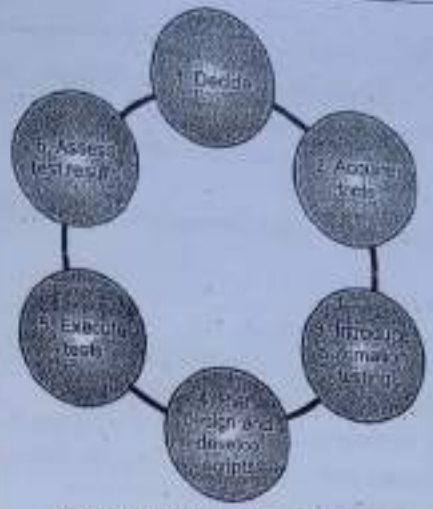


Fig. 6.4: Process of Automation Testing

Table 6.1: Difference between Manual Testing and Automation Testing

Sr. No.	Manual Testing	Automation Testing
1.	Test cases are executed by human resources so it is very slow and tedious.	Automation runs test cases significantly faster than human resources.
2.	As test cases need to be executed manually so more testers are required in manual testing.	Test cases are executed by using automation tool so less tester are required in automation testing.
3.	Manual testing is less reliable as tests may not be performed with precision each time because of human errors.	Automation tests perform precisely same operation each time they are run.
4.	No programming can be done to write sophisticated tests which fetch hidden information.	Testers can program sophisticated tests to bring out hidden information.
5.	It is inexpensive.	It is expensive.
6.	It gives low accuracy result.	It gives high accuracy result.
7.	It is done without interaction of any tool.	It is always done using tools.

- The various tools can be used for Automation Testing are HP Quick Test Professional, Selenium, IBM Rational Functional Tester, Silk Test, Test Complete, Testing Anywhere, Win Runner, Load Runner, Visual Studio Test Professional and so on.

6.3 VERIFICATION AND VALIDATION

- While doing the testing, the two terms Verification & Validation (V&V) have to be differentiated. Barry Boehm defines these terms based on the answer to the following questions:
  1. Verification (are we building the product right?)
  2. Validation (are we building the right product?)
- Validation is to check whether the software meets the customer expectations. Verification is to check whether the software conforms to specifications.
- In short, validation of the software is done to ensure that the software meets the requirements of the customer while verification of the software is done to ensure that the software meets the specifications.

1. Verification:

- Verification refers to a different set of authorities which ensure that software which has been built is traceable to customer requirements.
- Verification is a review without actually executing the process. Verification activities include testing and reviews.
- Verification implies code review and syntax check.

2. Validation:

- Validation is the process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements.
- Validation is checking the product with actual execution.

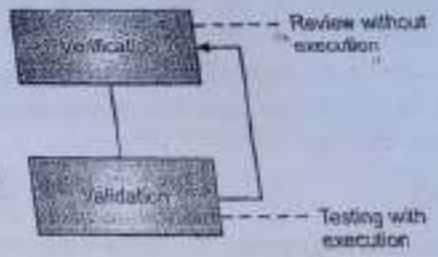


Fig. 6.5: Verification and Validation (V&V)

Table 6.2: Difference between Verification and Validation

Sr.No	Verification	Validation
1.	Verification is doing things right.	Validation is doing right things.
2.	It is process based.	It is product based.
3.	It ensures that the software system meets all the functionality.	It ensures that the functionalities meet the intended behavior.
4.	Verification takes place first and includes the checking for documentation, code etc.	Validation occurs after verification and mainly involves the checking of the overall product.
5.	Done by developers.	Done by testers.
6.	It has static activities, as it includes collecting reviews, walkthroughs, and inspections to verify a software.	It has dynamic activities, as it includes executing the software against the requirements.
7.	It is an objective process and no subjective decision should be needed to verify software.	It is a subjective process and involves subjective decisions on how well software works.

6.4 BLACK BOX TESTING AND WHITE BOX TESTING

- Once, the software is developed it should be tested in a proper manner before the system is delivered to the user. For this, two techniques that provide systematic guidance for designing tests are used. These techniques are:
  1. Once, the internal working of software is known, tests are performed to ensure that all internal operations of the software are performed according to specifications. This is referred to as **White Box Testing**.
  2. Once, the specified function for which the software has been designed is known, tests are performed to ensure that each function is working properly. This is referred to as **Black Box Testing**.

6.4.1 Black Box Testing

(BBA(CA)-S-18, W-18)

- **Black Box testing**, also known as Behavioral Testing , is a software testing method in which the internal structure/design/implementation of the item being tested is not known to the tester.
- Black Box (or functional) testing checks the functional requirements and examines the input and output data of these requirements.

- When Black Box testing is performed, only the sets of 'legal' input and corresponding outputs should be known to the tester and not the internal logic of the program to produce that output. Hence, to determine the functionality, the outputs produced for the given sets of input are observed.
- In Black Box testing, the tester is concentrating on what the software does, not how it does it.

Purpose of Black Box Testing:

1. To test the functionality of software.
2. Concerned with testing the specifications and does not ensure that all the components of software that are implemented are tested.
3. Addresses validity, behavior and performance of software.

- Black Box testing techniques attempts to find errors of the following categories.
  1. Incorrect or missing functions.
  2. Interface errors.
  3. Errors in data structures or external database access.
  4. Behavior or performance errors.
  5. Initialization and termination errors.

Example:

- A tester, without knowledge of the internal structures of a website, tests the web pages by using a browser, providing inputs (clicks, keystrokes) and verifying the outputs against the expected outcome.

Fig. 6.6: Black Box Testing

- Fig. 6.6 shows Black Box testing. A black-box test takes into account only the input and output of the software without regard to the internal code of the program.
- Black Box test cases are written first and white box test cases later. In order to write black box test cases, we need requirement document, design or project plan.
- Black Box testing is a testing strategy which is based solely on the requirements and specifications.
- This testing requires no knowledge of the internal paths, structure, or implementation of the software under test.
- In this testing, we check the overall functionality of the application.

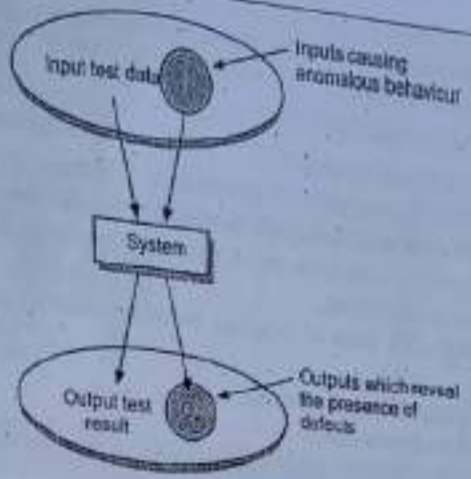


Fig. 6.7: Process of Black Box Testing

Techniques used in Black Box Testing:

- Following are some techniques that can be used for designing Black Box tests:
  1. **Equivalence Partitioning:** It is a software test design technique that involves dividing input values into valid and invalid partitions and selecting representative values from each partition as test data.
  2. **Boundary Value Analysis (BVA):** It is a software test design technique that involves determination of boundaries for input values and selecting values that are at the boundaries and just inside/ outside of the boundaries as test data.
  3. **Cause Effect Graphing:** It is a software test design technique that involves identifying the cases (input conditions) and effects (output conditions), producing a Cause Effect Graph, and generating test cases accordingly.

Advantages Black Box Testing:

1. In Black Box testing the code access is not required.
2. This testing is well suited and efficient for large code segments.
3. In this technique large numbers of moderately skilled testers can test the application without knowledge of implementation, programming language, or operating systems.
4. It clearly separates user's view from the developer's view through visibly defined roles.

Disadvantages Black Box Testing:

1. Inefficient testing, due to the fact that the tester only has limited knowledge about an application.
2. In this testing, the test cases are difficult to design.

3. Black Box testing has limited coverage, since only a selected number of test scenarios are actually performed.
4. Black Box testing blind coverage, since the tester cannot target specific code segments or error-prone areas.

6.4.2 White Box Testing

(BBA(CA)-S-III)

- White Box testing is also called Structural Testing and Glass Box Testing. White Box is a testing technique that takes into account the internal mechanism of a system or component.
- White Box test cases cannot be started in the initial phase of the project because it needs more architecture clarity which is not available at the start of the project.
- White Box is also known as structure-based testing technique because here the testers require knowledge of how the software is implemented, how it works.
- For example, a structural technique may be concerned with exercising loops in the software.

Purpose of White Box Testing:

1. To test the internal structure of software.
2. Test the software but does not ensure the complete implementation of all the specifications mentioned in user requirements.
3. Addresses flow and control structure of a program.

- **Example:** A tester, usually a developer as well, studies the implementation code of a certain field on a webpage, determines all legal (valid and invalid) and illegal inputs and verifies the outputs against the expected outcomes, which is also determined by studying the implementation code.

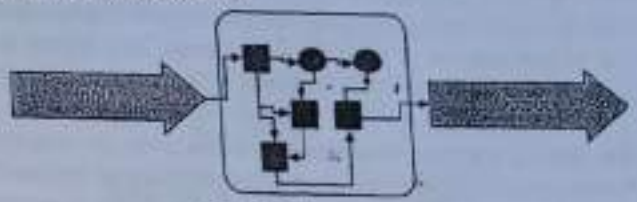


Fig. 6.8: White Box Testing

- White Box testing is the detailed investigation of internal logic and structure of the code.
- This Testing is done by developers as they know the internals of the application.
- In order to perform White Box testing on an application, a tester needs to know the internal workings of the code. The tester needs to have a look inside the source code and find out which unit/chunk of the code is behaving inappropriately.

- In white box, we do code reviews, view the architecture, and remove bad code practices and component level testing.
- Example of White Box testing and Black Box testing is shown in Fig. 6.9.

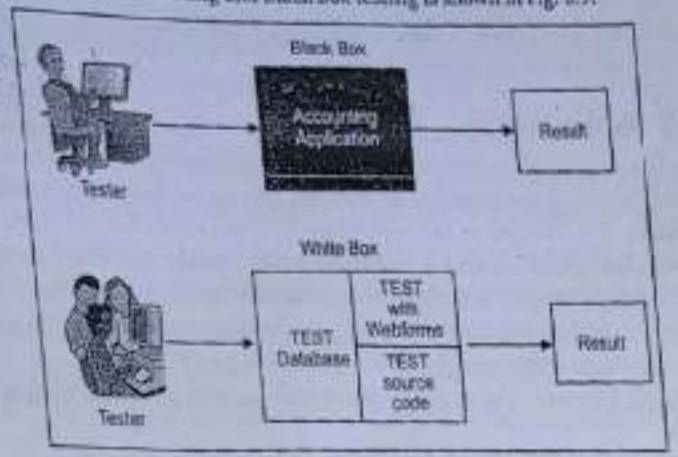


Fig. 6.9: White Box Testing and Black Box Testing

- White Box methods can be used for test generation and test adequacy analysis.
- These test cases are written after black box test cases are written.

Techniques used in White Box Testing:

1. **Basis Path Testing:** Basis path testing enables to generate test cases such that every path of the program has been exercised at least once. This technique is used to specify the basic set of execution paths that are required to execute all the statements present in the program.
2. **Path Testing:** A path through the program, which specifies a new condition or a minimum of one new set of processing statements. Path testing is where all possible paths through the code are defined and covered. It is a time consuming task.
3. **Data Flow Testing (DFT):** In this approach, tester tracks the specific variables through each possible calculations, thus defining the set of intermediate paths through the code. DFT tends to reflect dependencies but it is mainly through sequences of data manipulation. In short, each data variable is tracked and its use is verified. This approach tends to uncover bugs like variables used but not initialize, or declared but not used, and so on.
4. **Control Structure Testing:** Control structure testing is used to enhance the coverage area by testing various control structures (which include logical

structures and loops) present in the program. In condition testing, the test cases are derived to determine whether the logical conditions and decision statements are free from errors.

Advantages of White Box Testing:

1. It helps in optimizing the code
2. As the tester has knowledge of the source code, it becomes very easy and simple to find out which type of data can help in testing the application effectively.
3. Due to the tester's knowledge about the code, maximum coverage is attained during test scenario writing.
4. In this testing, extra lines of code can be removed which can bring in hidden defects.
5. White Box tests are easy to automate.

Disadvantages of White Box Testing:

1. It is difficult to maintain White Box testing, as it requires specialized tools like code analyzers and debugging tools.
2. Due to the fact that a skilled tester is needed to perform White Box testing, the costs are increased.
3. Sometimes, it is impossible to look into every nook and corner to find out hidden errors that may create problems, as many paths will go untested.

Table 6.3: Comparison between White Box Testing and Black Box Testing (BBA(CA)) S-25

Sr. No.	Black Box Testing	White Box Testing
1.	The internal workings of an application need not be known.	Tester has full knowledge of the internal workings of the application.
2.	Also, known as closed-box testing, data-drive testing, or functional testing.	Also, known as clear-box testing, structural testing, or code-based testing.
3.	Performed by end-users and also by testers and developers.	Normally done by testers and developers.
4.	Testing is based on external expectations - internal behavior of the application is unknown.	Internal workings are fully known and the tester can design test data accordingly.
5.	It is exhaustive and the least time-consuming.	The most exhaustive and time consuming type of testing.
6.	Not suited for algorithm testing.	Suited for algorithm testing.

Contd...

7.	This can only be done by trial-and-error method.	Data domains and internal boundaries can be better tested.
8.	Basic Logic Diagram: 	Basic Logic Diagram: 

6.5 UNIT TESTING

(BBA(CA): S-19)

- Unit testing involves the testing of each unit or an individual component of the software application. It is the first level of functional testing.
- The goal of unit testing is to isolate each part of the program and show that individual parts are correct in terms of requirements and functionality.
- A unit is a single testable part of a software system and tested during the development phase of the application software.
- Unit testing is performed by a selected group of software developers, unit testing involves evaluation of each unit, before it is integrated to for the system. These units are verified and validated on the basis of specified requirements, by testing them in isolation to identify, analyze and fix defects.
- Unit testing could be white-box, or black-box, or both.

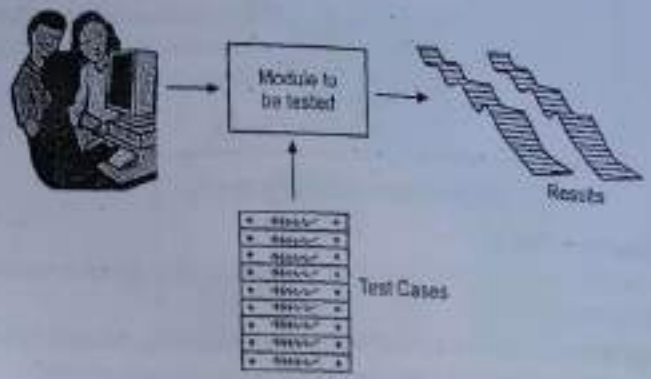


Fig. 6.10: Unit Testing Approach

Working of Unit Test:

- Whether performed manually or through automated tools, it follows a set process, which usually consist of the following activities:
- Defining and creating Unit Test Plans.
- Based on the specified test plans, unit test cases are designed and prepared.
- Execution of test cases on individual units.
- Fix the bugs (if any found) and re-evaluate the unit.
- The cycle of testing keeps on repeating, until the unit gets rid of all the bugs.

Concept of Stub and Driver:

- The most common approach to unit testing requires drivers and stubs to be written.
- Drivers and stubs are special-purpose arrangements, generally code, required to test units individually which can act as an input to the unit/module and can take output from unit/module. Driver and/or stub software must often be developed for each unit test.
- **Driver:** The driver simulates a calling unit and the stub simulates a called unit. A component is not a stand-alone program. In most applications, a driver is nothing more than a "main program" that accepts test case data, passes such data to the component (to be tested), and prints relevant results.
- **Stubs:** Stubs serve to replace modules that are subordinate (invoked by) the component to be tested. A stub or "dummy sub-program" may do minimal data manipulation, prints verification of entry, and returns control to the module undergoing testing.
- Driver and stubs are software that must be written (formal design is not commonly applied) but that is not delivered with the final software product.
- Fig. 6.11 shows stub and driver concept in unit testing.

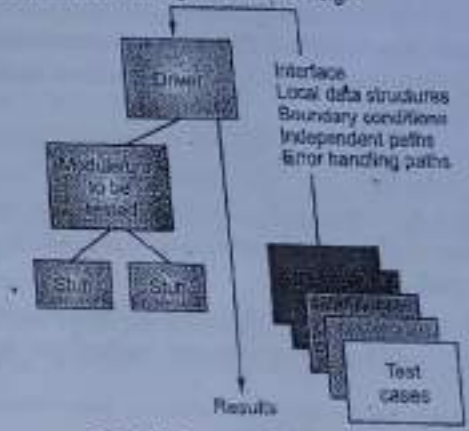


Fig. 6.11: Concept of Stub and Driver

dvantages:

1. The bugs are found early in unit testing hence it helps in reducing the cost of bug fixes.
2. Unit testing helps in simplifying the debugging process. If suppose a test fails then only latest changes made in code needs to be debugged.
3. Unit testing also helps in maintaining and changing the code. This is possible by making the codes less interdependent so that unit testing can be executed. Hence, chances of impact of changes to any other code get reduced.

Disadvantages of Unit Testing :

1. Unit testing cannot catch each and every bug in a software application.
2. It cannot evaluate every execution path.
3. There is a limit to the number of scenarios and test data that a developer can use to verify a source code. After having exhausted all the options, there is no choice but to stop unit testing and merge the code segment with other units.

Unit Testing Tools:

- Unit Testing may be manual but there are several Unit Testing Frameworks/Tools used to create accurate unit tests. We will see some of them :
  - **JUnit:** JUnit is an open-source unit testing framework designed for Java Programming Language. Supportive for the test-driven environment. Test data is first tested and then inserted in the piece of code. It provides annotation for test method identification, an assertion for testing expected results and test runners. It is simplest tool and helps to write code easily and faster.
  - **NUnit:** NUnit is a unit testing framework based on.NET platform. It is a free tool allows to write test scripts manually but not automatically. It works in the same way as JUnit works for Java. It supports data-driven tests that can run in parallel.
  - **PHPUnit:** PHPUnit is a unit testing tool for PHP programmer. It takes small portions of code which is called units and tests each of them separately. This tool also allows developers to use pre-define assertion methods to assert that a system behave in a certain manner.
  - **JMockit:** JMockit is an open-source tool for Unit Testing with the collection of tools and API. Developers can use these tools and API to write test using TestNG or JUnit. It is considered as an alternative to the conventional use of the mock object. This tool provides 3 types of code coverage such as Line Coverage, Path Coverage, and Data Coverage.
  - **EMMA:** It is an open-source toolkit for analyzing and reporting code written in a Java language. EMMA support coverage types like method, line, basic block. It is a Java-based tool.

6.6 INTEGRATION TESTING (BBA(CA)-W.U. 15)

- As the components are constructed and tested then they are linked together to check if they work with each other. It is a fact that two components that have passed all their tests, when connected to each other produce one new component full of faults. These tests can be done by specialists, or by the developers.
- Integration testing is focused on how they communicate with each other, as specified in the "system design (defines relationships between components)".
- The tests are organized to check all the interfaces, until all the components have been built and interfaced to each other producing the whole system.
- Integration testing is defined as, "the testing of combined parts of an application to determine if they function correctly".

Working of Integration Testing:

- Once, unit testing is complete, integration testing begins. In integration testing, the units validated during unit testing are combined to form a subsystem.
- The integration testing is aimed at ensuring that all the modules work properly as per the user requirements when they are put together i.e. integrated.
- The objective of integration testing is to take all the tested individual modules, integrate them, test them again, and develop the software, which is according to design specifications.

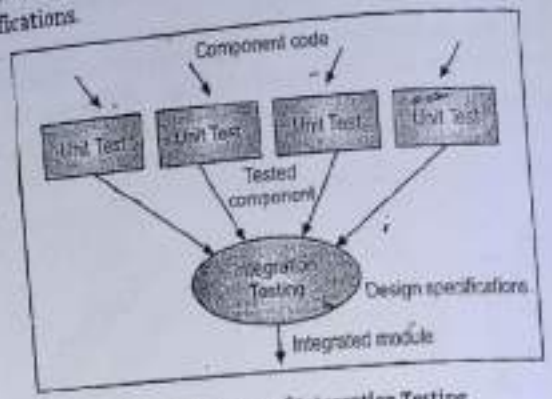


Fig. 6.12: Concept of Integration Testing

Approaches of Integration Testing:

- Integration testing can be done in two ways i.e., Bottom-up Integration testing a Top-down Integration testing.

1. **Bottom-up Integration:** This testing begins with unit testing, followed by tests progressively higher-level combinations of units called modules or builds.

2. **Top-down Integration:** In this testing, the highest-level modules are tested first and progressively, lower-level modules are tested thereafter.

- In a comprehensive software development environment, bottom-up testing is usually done first, followed by top-down testing. The process concludes with multiple tests of the complete application, preferably in scenarios designed to mimic actual situations.

Advantages of Integration Testing:

1. The unit modules may not be tested as per the changes made to the requirements that will lead to error page when Integration Testing is done.
2. It is easy to fix the error in the Integration Testing when compared to the System testing.
3. The interfaces should be checked thoroughly if any error messages are shown.

Types of Integration Testing:

Fig 6.13: Types of Integration Testing

- Let us learn the types of integration testing in detail.

1. **Top-down Integration:**

- In this integration, modules are integrated by moving downward through the control hierarchy beginning with the main module.
- Subordinate modules are incorporated in either a Depth-First (DF) or Breadth-First (BF) fashion.

(i) **DF:** All modules on a major control path are integrated.
(ii) **BF:** All modules directly subordinate at each level are integrated.

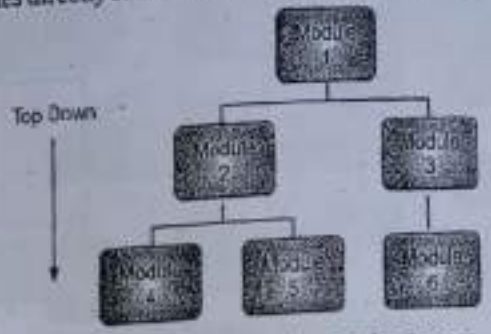


Fig. 6.14: Top-down Integration Testing Process

Advantages:

(i) The tested product is very consistent because the integration testing is basically performed in an environment that almost similar to that of reality.
(ii) This approach verifies major control or decision points early in the test process.
(iii) Stubs can be written with lesser time as compared to the drivers.

Disadvantages:

(i) Stubs need to be created to substitute for modules that have not been built or tested yet and this code is later discarded.
(ii) Because stubs are used to replace lower level modules. No significant data flow can occur until much later in the integration/testing process.
(iii) In this approach the basic functionality is tested at the end of cycle.

2. **Bottom-up Integration:**

- Integration and testing starts with the most atomic modules in the control hierarchy.

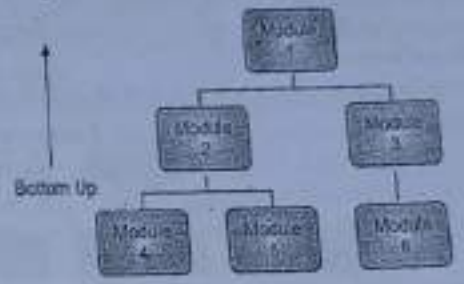


Fig. 6.15: Bottom-up Integration Testing Process

Advantages:

(i) This approach verifies low-level data processing early in the testing process.
(ii) In this approach, need for stub is eliminated.
(iii) In this approach, development and testing can be done together so that the product or application will be efficient and as per the customer specifications.

Disadvantages:

(i) Driver modules need to be built to test the lower-level modules for this code is later discarded or expanded into a full-featured version.
(ii) Drivers inherently do not contain the complete algorithms that will eventually use the services of the lower-level modules; consequently, testing may be incomplete or more testing may be needed later when the upper level modules are available.

3. **Sandwich Integration:**

- Sandwich integration also referred as Mixed Integration or Bi-directional Integration
- Proceeds using functional groups of modules, with each group completed before t next.

- Sandwich testing defines testing into two parts and follows both parts starting from both ends, i.e., top-down approach and bottom-up approach either simultaneously or one after another.
- In top-down approach, testing can start only after the top-level modules have been coded and unit tested. Similarly, bottom-up testing can start only after the bottom level modules are ready.
- Sandwich approach overcomes this shortcoming of top-down and bottom-up approaches.
- In sandwich integration approach, testing can start as and when modules become available. Therefore, this is one of the most commonly used integration testing approach.
- Sandwich integration testing is also a vertical incremental testing strategy that tests the bottom layers and top layers and tests the integrated system in the computer software development process.
- Using stubs, it tests the user interface in isolation as well as tests the lower level functions using drivers.

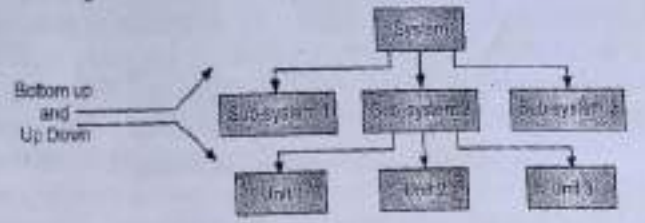


Fig. 6.16: Sandwich Testing Approach

Advantages of Sandwich Integration Testing:

(i) In this testing, both top-down and bottom-up approaches start at a time as per development schedule. Units are tested and brought together to make a system. Integration is done downwards.

(ii) This approach is useful for very large projects having several sub-projects. When development follows a spiral model and the module itself is as large as a system.

Disadvantages of Sandwich Integration Testing:

(i) It cannot be used for smaller systems with huge interdependence between different modules.

(ii) The costs of this type of testing are quite high as both the approaches are used in the completion of testing.

4. Incremental Integration:

- Incremental integration testing encompasses the basic concepts of integration testing.

- It is continuous or repetitive testing of a software application as new and fresh functionality is advised.
- The incremental integration approach tests program in small increments. It is easier to detect errors in incremental approach because only a small segment of software code is tested at a given instance of time. Moreover, interfaces can be tested completely if this approach is used.
- There are complex interactions between system components and, when an anomalous output is discovered, we may find it hard to identify where the error occurred.
- Initially, we should integrate a minimal system configuration and test this system. Then add components to this minimal configuration and test after each added increment.

Working of Incremental Integration:

- In incremental integration type, we start with one module and unit test it. Then combine the module which has to be merged with it and perform test on both the modules.
- In this way, incrementally keep on adding the modules and test the recent environment. Thus, an integrated tested software system is achieved.
- In the Fig. 6.17, A, B, C and D are components and T1 to T5 are related sets of tests of the features incorporated in the system. T1, T2 and T3 are first run on a system composed of component A and component B, (the minimal system).

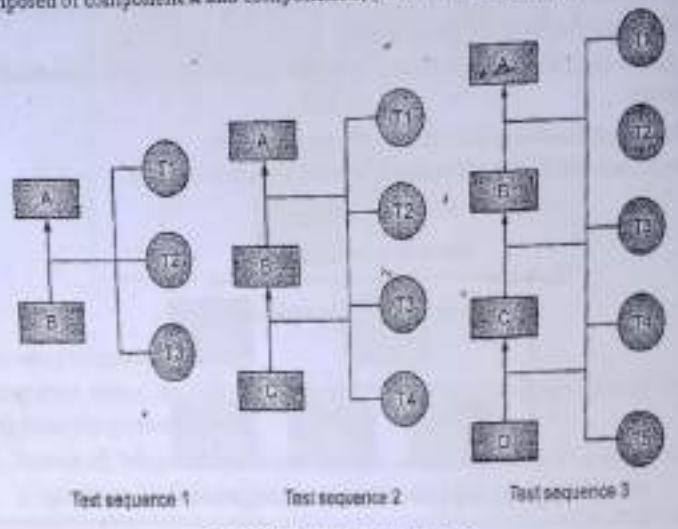


Fig. 6.17: Incremental Integration

- If these are reveal defects, they are corrected. Component C is integrated and T1, T2 and T3 are repeated to ensure that there have not been unexpected interactions with A and B. If problems arise in these tests, this probably means that they are due to interactions with the new component.
- The source of the problem is localized, thus simplifying defect location and repair. Test set T4 is also run on the system. Finally, component D is integrated and tested using existing and new tests (T5).
- Incremental integration testing is beneficial for the following reasons:
  - (i) Incremental integration approach does not require many drivers and stubs.
  - (ii) It is easy to localize the errors since modules are combined one by one.
  - (iii) The first suspect is the recently added module. Thus, debugging becomes easy and simple.

5. Non-incremental Integration:

- The non incremental approach is also known as "Big-Bang Integration Testing".
- Big-bang integration approach is the most commonly seen approach at many places where the system is tested completed after development is over. There is no testing of individual units/modules and integration sequence.
- Big Bang Integration Testing is an approach in which all software components (modules) are combined at once and make a complicated system. This unity of different modules is then tested as an entity. According to this checking method, the integration process will not be executed until all components are completed.
- When this type of testing strategy is adopted, it is difficult to isolate any errors found, because attention is not paid to verifying the interfaces across individual units.
- Fig. 6.18 shows the Big Bang testing workflow:

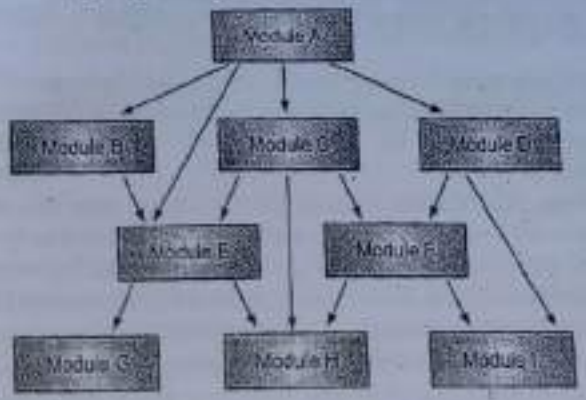


Fig. 6.18: The Big Bang testing workflow

Advantages of Big-bang Integration:

- (i) In this integration, no stub/driver is required to be designed and coded.
- (ii) In this integration, the cost involve is very less/low as it does not involved much creation of test artifacts. Sometimes, test plan is also not created.
- (iii) Big-bang integration approach is very fast.
- (iv) It suitable for small systems.

Disadvantages of Big-bang Integration:

- (i) In this testing, the location of defects may not be found easily.
- (ii) Problems found in this approach are hard to debug.
- (iii) Failures occur more frequently because of the simultaneous check of numerous modules.
- (iv) A single mistake can influence the results of the whole integration testing.

Approaches of Integration Testing:

1. **Top-down Testing:** Each module or class from the top level of program segment and after that we can add the first one to next and continue this process until complete module check.
2. **Bottom-up Testing:** It is a reserve case of top down testing but a simple difference is that this does not allow to integration the tested modules from bottom up.
3. **Regression Testing:** Defines and examine with some existing function and future achievement goal in specific software regression testing allows the software development team to ensure that the existing functions and other modules whether they can use for their coming project or not.
4. **Smoke Testing:** It defines the testing approach where we can implement and test each and every module and function with the existing project. This type of testing approach is basically needed when we need to build up a long-term project.

- **Integration Testing Tools:** There are several tools available for this testing. Some of them are: Rational Integration Tester, Protractor, Steam, TESSY.

Table 6.4: Difference between Unit testing and Integration Testing

Sr. No.	Unit Testing	Integration Testing
1.	Validates the system unit-by-unit.	Assesses the system as a whole by integrating several modules simultaneously.
2.	Unit testing mainly focus on the testing the functionality of individual units only and does not uncover the issues arises when different modules are interacting with each other.	Integration testing is to be carried out to discover the issues arise when different modules are interacting with each other to build overall system.

3.	Unit testing is not further sub divided into different types	Integration testing is further divided into different types as follows: Top-down Integration, Bottom-Up Integration and so on.
4.	First level of software testing.	This type of testing is carried out after Unit testing and before System testing.
5.	Unit tests should have no dependencies on code outside the unit tested.	Strongly relies on dependencies, involves the use of databases.
6.	Tests are faster to perform.	Tests are slower.
7.	Usually performed by developers.	Requires an experienced testing team.
8.	This is performed at the coding level.	This is performed at the communication level.

6.7 SYSTEM TESTING [BBA/CA]:W-18]

- IEEE defines system testing as, "a testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirement"
- System testing during development: involves integrating components to create a version of the system and then testing that integrated system.
- System testing checks that components are compatible, interact correctly and transfer the right data at the right time across their interfaces.
- Components developed by different team members or groups may be integrated at this stage.
- System testing may involve a separate testing team with no involvement from designers and programmers.
- System testing should focus on testing the interactions between the components and objects that make up a system. We may also test reusable components or systems to check that they work as expected when they are integrated with new components.
- Interaction testing also helps find misunderstandings, made by component developers, about other components in the system. Because of its focus on interactions, use case-based testing is an effective approach to system testing.
- Typically, each use case is implemented by several components or objects in the system.
- System testing is a collective rather than an individual process. That means System testing is not about checking the individual parts of the design, but about checking the system as a whole.

- System testing is important because of the following reasons:
1. System testing is the first step in the Software Development Life Cycle, where the application is tested as a whole.
2. The application is tested thoroughly to verify that it meets the functional and technical specifications.

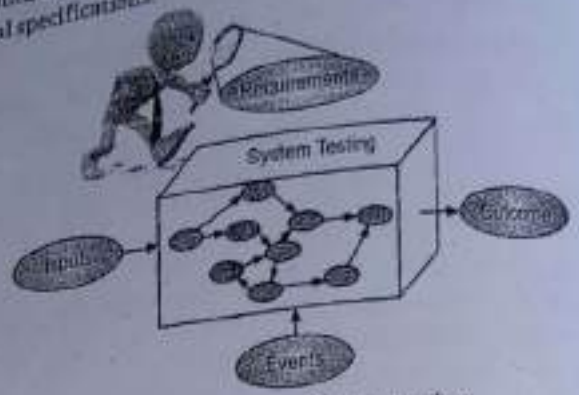


Fig. 6.19: Concept of System Testing

Working Process of System Testing:

- There are six normal steps that are a part of system testing process which are explained below:

1. **Make a Test Plan:** The first step of the procedure includes test plan creation, wherein the lead or test supervisor characterize the scope and target of testing, decides the techniques, settles on manual and automated testing, characterize the exit and entry criteria, assigns responsibilities and roles, in addition to other things.
2. **Test Case Writing:** From this step, the execution of testing is started by the team. Test cases are set up based on use cases and the requirements of testing as well as the customer/user, like UI, technical, non-functional, functional and performance, etc.
3. **Select Test Data:** Once the test cases are created by the team, they co-operate to choose or make the required test data, which plays a vital role in test execution. These are the input data sources that help the testers get anticipated outcomes.
4. **Test Case Execution:** Finally, the test cases made earlier are executed by the team who always monitor the procedure and record any errors or issues experienced by them during the procedure. Likewise, the outcome of the testing is additionally recorded here.
5. **Bug Reporting and Fixation:** In this phase of the procedure that the team reports all the recorded bugs and issues to the concerned individual from the team. When revealed, the software engineer or the developer works with the testing team to fix and resolve the issue.

6. **Repeat the Test Cycle (If required):** After all of the issues and bugs are fixed, the testers repeat the test cycle to get the expected outcomes.

Advantages:

1. System tests help clearly specify how the application should behave.
2. System tests can be run automatically (for example, each night) so that testing is done as the application is being developed.
3. System tests help us to test that the application is working correctly from the point of view of a user.
4. System tests help us to test that changes made to one part of the application have not created a bug somewhere else.

Disadvantages:

1. System testing starts late when all the components are ready. Hence, the cost of fixing bugs is higher.
2. The localization of the bugs is difficult as the entire system is participating in the testing.

Types of System Testing:

- System Testing is called a superset of all types of testing as all the major types of testing are covered in it. Types of testing may vary on the basis of product, organization processes, timeline, and requirements. Some of the common System testing types are:
  - (i) **Functionality Testing:** It is also known as 'functional completeness testing'. To ensure that the functionality of the product working as per the requirements defined and within the capabilities of the system.
  - (ii) **Recoverability Testing:** To ensure how well the system recovers from various input errors and other failure situations. Recovery testing done to show software solution is reliable, trustworthy and successfully recover from possible crashes.
  - (iii) **Interoperability Testing:** To ensure whether the system works well with third-party products or not.
  - (iv) **Performance Testing:** To make sure the system's performance under the various condition, regarding performance characteristics.
  - (v) **Regression Testing:** To make sure the system's stability as it passes through the integration of different subsystems and maintenance tasks. It also makes sure no old bugs appear from the addition of new software modules over time.
  - (vi) **Usability Testing:** To make sure the user's ease to use the application, flexibility in handling controls and the ability of the system to meet its objectives.
  - (vii) **Documentation Testing:** To make sure that the system's user guide and other help topics documents are correct and usable.
  - (viii) **Security Testing:** To make sure that the system does not allow unauthorized access to data and resources.

- **Smoke Testing:** Smoke testing, also known as 'Build Verification Testing', is a type of software testing that comprises of a non-exhaustive set of tests that aim at ensuring that the most important functions work. The results of this testing are used to decide if a build is stable enough to proceed with further testing.
- **Some more types of software testing are:** Scalability Testing, Reliability Testing, Graphical User Interface Testing (GUI), Exception Handling, Volume Testing, Stress Testing, Sanity Testing, Exploratory Testing, Ad-hoc Testing, Installation Testing, Maintenance of Testing

Summary

- Software testing is the process of executing a software program or an application for finding the bugs.
- An error is a human action that produces an incorrect result.
- A defect is a flaw in a component or system that can cause the component or system to fail to perform its required function, e.g. an incorrect statement or data definition. A defect, if encountered during execution, may cause a failure of the component or system.
- A failure is actual deviation of the component or system from its expected delivery, service or result.
- A software bug is an error, flaw, failure or fault in a computer program or system that causes it to produce an incorrect or unexpected result, or to behave in unintended ways.
- A test case is a set of conditions or variables under which a tester will determine whether a system under test satisfies requirements or works correctly.
- A test plan is a document describing the scope, approach, resources and schedule of intended test activities. It identifies amongst others test items, the features to be tested, the testing tasks, who will do each task, degree of tester independence, the test environment, the test design techniques and entry and exit criteria to be used, and the rationale for their choice, and any risks requiring contingency planning. It is a record of the test planning process.
- Software Testing Life Cycle (STLC) refers to a testing process which has specific steps to be executed in a definite sequence to ensure that the quality goals have been met. In STLC process, each activity is carried out in a planned and systematic way. Each phase has different goals and deliverables.
- Testing of software can be done in both Automation testing and Manual testing method. The method takes automation tool support to execute the test cases is known as Automation Testing. Automated testing is good for large project. Manual testing is a method used by software developers to run tests manually.
- In Unit Testing to test specific component of software or module. It is specially done by programmers and not by testers, because it needs thorough knowledge of the internal programming design and code.

- Unit testing is performed at developer's end to make sure that their code is working well and meets the user specifications. The aim of unit testing is to divide every part of the program and test that the separate part are working correctly.
- In unit testing the Stub is considered as subprogram. Driver is a simple main program.
- Black Box testing is a testing technique to test functionalities and requirements of the system. It does not test the internal part of the system.
- White Box testing a testing technique based on information of the internal logic of an application's code and also known as Glass box Testing. It works on internal working code of the system. Tests are based on coverage of code statements, branches, paths, conditions.
- Integration testing is a technique to verify joint functionality after integration. Modules are typically code modules, individual applications, client and server applications on a network, etc. This type of testing is especially applicable to client/server and distributed systems.
- Some different types of integration testing are big-bang, top-down, and bottom-up and mixed (sandwich).
- Bottom-up testing is an approach to integrated testing where the lowest level components are tested first, then used to facilitate the testing of higher level components. The process is repeated until the component at the top of the hierarchy is tested.
- Top-down testing is an approach to integrated testing where the top integrated modules are tested and the branch of the module is tested step by step until the end of the related module.
- Sandwich testing is an approach to combine top-down testing with bottom-up testing.
- In the big-bang approach, most of the developed modules are coupled together to form a complete software system or major part of the system and then used for integration testing. This method is very effective for saving time in the integration testing process.
- System testing is a technique to test whole system.
- Acceptance testing is a type of testing verifies that the system meets the customer specified requirements or not. User or a customer does this testing to decide whether to accept application.
- Specification-based testing technique is called as Black Box or Functional testing technique.
- In equivalence partitioning, software testing technique divides the input data of a software unit into the partitions of equivalent data and test cases are derived from the partitions of equivalent data.
- Boundary Value Analysis (BVA) is the test case design technique to test boundary value between partitions. Boundary value is an input or value on the border of an equivalence partition. It consists of start-end, lower-upper, maximum-minimum on inside and outside boundaries.
- Structure based technique is called as White box testing or Glass Box testing.

Check Your Understanding

1. Which test is also known as structural testing?
   (a) Black box testing (b) White box testing
   (c) Grey box testing (d) System testing
2. Which one of the following technique is applied for usability testing?
   (a) Black box testing (b) White box testing
   (c) Grey box testing (d) System testing
3. Verification is _____.
   (a) Making sure that it is what the user really wants.
   (b) Checking that we are building the right product.
   (c) Checking that we are building the product right.
   (d) Performed by an independent test team.
4. Software mistakes during coding are known as _____.
   (a) Errors (b) Bugs
   (c) Failures (d) Defects
5. Which test requires drivers and stubs to be written?
   (a) Acceptance testing (b) Smoke testing
   (c) Unit testing (d) Integration testing

ANSWER KEY

1. (b)	2. (a)	3. (c)	4. (b)	5. (c)

Practice Questions

Q.I: Answer the following Questions in short.

1. What is software testing?
2. Which are unit testing tools?
3. Enlist objectives of software testing.
4. Describe need of software testing.
5. What is Black Box testing?

Q.II: Answer the following Questions.

1. With the help of diagram describe White Box testing.
2. List the advantages and disadvantages of Black Box testing and White Box testing.
3. Explain Integration testing in detail.
4. Write Difference between:
   (a) Verification and Validation
   (b) Unit Testing and Integration Testing
   (c) Manual and Automation Testing
5. Write types of system testing?

6. Which are advantages of system testing?
7. Write a short note : Unit testing.
8. Explain PDCA cycle of software.

Q.III: Define the following terms.

1. Functional Testing
2. White Box Testing
3. Unit Testing
4. Structure Chart
5. Defect
6. Top-Down Approach
7. Test Case
8. Test Plan
9. Stub And Driver
10. Unit

vious Exams Questions

BBA (CA)

Summer 2018

1. What is Software testing ?

Ans. Refer to Section 6.1 [2M]

2. What is Black Box Testing ? Explain the methods used in BBT.

Ans. Refer to Section 6.4.1 [2M]

3. Write short note : White Box Testing.

Ans. Refer to Section 6.4.2 [4M]

Winter 2018

1. What is integration testing ?

Ans. Refer to Section 6.6. [2M]

2. Explain testing principles and objectives.

Ans. Refer to Section 6.1 [4M]

3. Write short note: Black Box Testing

Ans. Refer to Section 6.4.1 [4M]

Summer 2019

1. Define unit testing.

Ans. Refer to Section 6.5 [2M]

2. Differentiate between white-box and Black-box testing.

Ans. Refer to Section 6.4.2. [4M]

❖❖❖

7... Software Maintenance & Software Re-engineering

Objectives...

- To understand concepts of Software Maintenance and Reengineering
- To get information about Reverse Engineering
- To learn concept of Restructuring and Forward Engineering

7.1 INTRODUCTION

- Software maintenance is one of the principal aspect of software engineering. The software maintenance is a set of software engineering activities that occur after the software has been delivered to the end-user.
- The need for software maintenance arises due to changes required in the software system.
- Reengineering denotes a methodology that addresses the analysis of existing software systems and their reuse, complete or in parts, to build adapted systems where the purpose or the construction of the new systems differs to a great extent.
- Reverse engineering is the process of discovering the functions and their inter-relationship of a software system as well as creating representations of the system in another form or at a higher level of abstraction.
- In this chapter, we will study about Software Maintenance, Re-engineering, Forward Engineering with their basic concepts.

7.2 SOFTWARE MAINTENANCE DEFINITION AND TYPES

- Software systems evolve over time. The evolution has several reasons and different scales during the software life cycle.

- Once the software is delivered and deployed, its intended users will begin to report the newly discovered errors. They will also demand additions and modifications to reflect new requirements or operating environments. Software maintenance includes all activities related to software enhancements performed after the software is deployed.

7.2.1 Definition of Software Maintenance

- Software maintenance stands for all the modifications and updates done after the delivery of software products.
- Failures continue to be discovered in software for years. Software maintenance is the set of activities, both technical and managerial, that ensures that software continues to meet organizational and business objectives in a cost effective way.
- Software maintenance is defined as, "the totality of activities required to provide cost-effective support to software activities are performed during the pre-delivery stage as well as during the post-delivery stage as explained below:
  1. **Pre-delivery activities** include planning for post-delivery operations, maintainability, and logistics determination for transition activities.
  2. **Post-delivery activities** include software modification, training, and operating or interfacing to a help desk."

OR

- Software maintenance is defined as, "the process of modifying a software system or component after delivery to correct faults, improve performances or other attributes, or adapt to a changed environment".
- Five factors drive the difficulty of delivering software i.e. input vs. output, cycle-time, cost/change, schedule and flexibility.
- Maintenance is the act of keeping an entity in an existing state of efficiency, validity, to preserve from failure or decline. For Example., car maintenance.
- Maintenance keeps the software up-to-date with environment changes and changing user requirements. The change might be simple changes to correct coding errors, more extensive changes to correct design errors.

Examples of Software Maintenance:

- **Y2K:** Y2K problem is an interesting example of the role of maintenance in software. (Y2K was the need to fix software that handled years as two-digit numbers when the calendar switched from 1999 [99] to 2000 [00]. Some programs took that to mean that time had moved backwards.)
- **Operating System Patching:** Microsoft, Apple, Linux/Unix. OS is core to use of computer, so it must be constantly maintained.
- **Commercial Software in General:** Customers need to be informed of updates. Updates have to be easily available.
- **Anti-Virus Software:** Do not usually have to update software, but send virus definitions.

Need for Software Maintenance:

- New-a-days, software maintenance is widely accepted part of the SDLC. It stands for all the modifications and updations done after the delivery of software products. There are a number of reasons, why modifications are required, some of them are mentioned below:
  1. **Continuous Change:** The environment in which the software operates keeps on changing, therefore, the software must also be changed to work in the new environment.
  2. **Increasing Software Complexity:** The structure of the software becomes more complex with continuous change in software, therefore, some preventive steps must be taken to improve and simplify its structure.
  3. **Large and Latest (New) Software Evolution:** Software evolution is a self-regulating process. Software attributes such as size, time between releases, and the number of reported errors are almost constant for each system release.
  4. **Organizational Stability:** The rate with which the software is developed remains approximately constant and is independent of the resources devoted to the software development.
  5. **Market Conditions:** Policies, which changes over the time, such as taxation and newly introduced constraints like, how to maintain book-keeping may trigger need for modification.
  6. **Client Requirements:** Over the time, customer may ask for new features or functions in the software.
  7. **Host Modifications:** If any of the hardware and/or platform (such as operating system) of the target host changes, software changes are needed to keep adaptability.
  8. **Organization Changes:** If there is any business level change at client end, such as reduction of organization strength, acquiring another company, organization venturing into new business, need to modify in the original software may arise.

7.2.2 Types of Software Maintenance

- In a software lifetime, type of maintenance may vary based on its nature. It may be just a routine maintenance tasks as some bug discovered by some user or it may be large event in itself based on maintenance size or nature.

Following are types of maintenance based on their characteristics:

1. **Corrective Maintenance:**

- This type of maintenance includes modifications and updations done in order to correct or fix problems. These are either discovered by user or concluded by user error reports.
- In other words, it is the process of identifying, isolating and repairing the faults (errors) that are discovered in the system so that the system can be restored and operational.
- Corrective maintenance is concerned with fixing errors that are observed when the software is in use.
- Corrective maintenance happens after the fault. Faults (errors) can be user based, coding (software) based, or hardware based.
- Corrective maintenance is 'traditional maintenance' while the other types are considered as 'software evolution.'
- Types of corrective maintenance are explained below:
  - (i) **"Deferred corrective maintenance"**, (in which work is delayed in conformance to stakeholders), and
  - (ii) **"Immediate corrective maintenance"**, (actual work starts immediately after failure and is critical in nature).

2. **Adaptive Maintenance:**

- This type of maintenance includes modifications and updations applied to keep the software product up-to -date and tuned to the ever-changing world of technology and business environment.
- Adjusting the system adaptively based on the environment changes is called as adaptive maintenance.
- They system has no faults (errors) on its own, but needs adapting (changing) as per the environment changes.

Examples:

- Healthcare related systems need to be updated during protocol changes by the Health Insurance Portability and Accountability Act (HIPAA).
- Adjusting the system parameters as per the new hardware changes/OS changes.
- A maintenance program undertaken to make the website compatible with the new version of the software is an example of adaptive maintenance. Fig. 7.1 shows a website got upgraded once the newer version of the internet explorer browser was released.

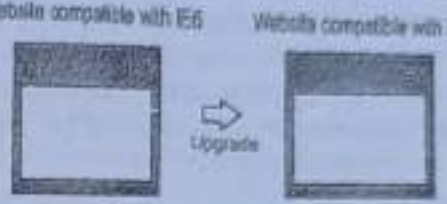


Fig. 7.1: Example of Adaptive Maintenance

3. **Perfective Maintenance:**

- This type of maintenance includes modifications and updates done in order to keep the software usable over long period of time. It includes new features, new user requirements for refining the software and improve its reliability and performance.
- This includes enhancing both the function and efficiency of the code and changing the functionalities of the system as per the users' changing needs.
- Examples of perfective maintenance:
  - a. Modifying the payroll program to incorporate a new union settlement and adding a new report in the sales analysis system.
  - b. Upgrading hardware, software, and bandwidths
  - c. The older version of the keyboards did not have the multimedia keys (See Fig. 7.2). But with the popularization of multimedia applications, it became necessary to provide the users some short-cut keys for accessing the multimedia functionalities directly from the keyboard.

Fig. 7.2: Example of Perfective Maintenance

- Perfective maintenance is the system upgrade kind of work which makes the system work more efficiently. About 50% of the maintenance is perfective maintenance.

4. **Preventive Maintenance:**

- This type of maintenance includes modifications and updations to prevent future problems of the software. It aims to attend problems, which are not significant at this moment but may cause serious issues in future.

- Preventive maintenance comprises documentation updating, code optimization, and code restructuring. Documentation updating involves modifying the documents affected by the changes in order to correspond to the present state of the system. Code optimization involves modifying the programs for faster execution or efficient use of storage space.
- In other words, Activities that are aimed at increasing the maintainability are called as preventive maintenance.
- **Examples of Preventive maintenance :**
  - In automobile, the car's engine oil gets changed every 5000 km to prevent errors.
  - Documenting the existing system.
  - Identifying system component that may fail depending on the trend.
- Fig. 7.3 shows for maintenance type ratios. It clearly indicates perfective maintenance is more than any type and it contributes 50%. Only 4% of maintenance types are preventive maintenance.

Fig. 7.3: Chart to show Maintenance Type Ratios

7.2.3 Cost of Maintenance

- Reports in real-world software environment suggest that the cost of maintenance is high. A study on estimating software maintenance found that the cost of maintenance is as high as 67% of the cost of entire software process cycle.
- On an average, the cost of software maintenance is more than 50% of all SDLC phases. There are various factors, which trigger maintenance cost go high, such as:

Factors affecting Maintenance Cost:

- Factors of Maintenance cost are divided into two subcategories:

1. **Non-Technical factors:**
   The Non-Technical factors include:

- **Application Domain:** If the application of the program is defined and well understood, the system requirements may be definitive and maintenance due to changing needs minimized.
- **Team stability:** Maintenance cost are reduced if same staff are involved with them for some time.
- **Program lifetime:** If the program becomes outdated, or their original hardware is replaced, then conversion costs exceed rewriting costs.
- **Dependence on External Environment:** If an application is dependent on its external environment, it must be modified as the situation changes.
- **Hardware stability:** Maintenance cost not required if an application is designed to operate on a specific hardware configuration and that configuration does not changes during the program's lifetime.

2. **Technical factors:**
   Technical factors include the following:
   - **Module independence:** It should be possible to change one program module of a system without affecting any other module.
   - **Programming language:** Programs written in a high-level programming language are generally easier to understand than programs written in a low-level language.
   - **Programming method:** The programming method contributes to its understandability and so, it must easy to modify and understand.
   - **Program validation and testing:** There may be fewer bugs in the program if more time and efforts are spent on design validation and program testing. So, maintenance costs resulting from bugs correction are lower.
   - **Documentation:** If a program is supported by clear, complete however short documentation, the functions of understanding the application can be associatively straight-forward.
   - **Configuration management techniques:** One of the essential costs of maintenance is keeping track of all system documents .This includes configuration identification, change control, status reports, the configuration of the program, version management, configuration auditing. The maintenance costs get reduced by effective control of software configuration management technology that strengthens and maintain the management of the maintenance work.

7.2.4 Maintenance Activities

- IEEE provides a framework for sequential maintenance process activities. It can be used in iterative manner and can be extended so that customized items and processes can be included.

- Fig. 7.4 shows various maintenance activities.

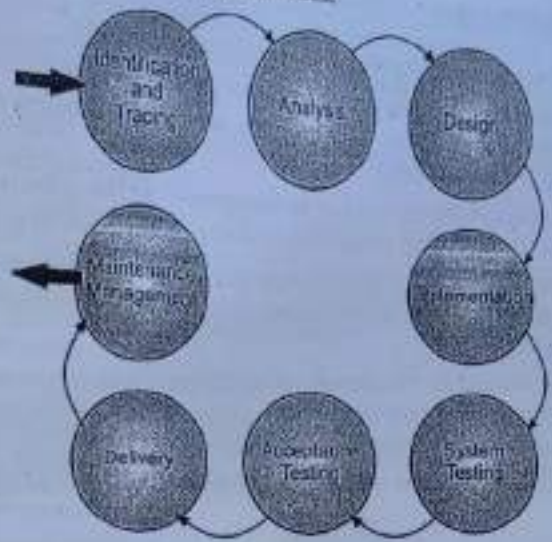


Fig. 7.4: Maintenance Activities

- These activities go hand-in-hand with each of the following phase:
  1. **Identification and Tracing:** In this step, the request for modifications in the software is identified and the maintenance type is also classified.
  2. **Analysis:** In this step, the scopes of each validated modification request is determined and a plan is prepared to incorporate the changes in the software.
  3. **Design:** In this step, the modifications to be made in the software are designed. The new modules, which need to be replaced or modified, are designed against requirement specifications set in the previous stage. Test cases are created for validation and verification.
  4. **Implementation:** In this step, the actual modifications in the software code are made, new features that support the specification of present software are added, and the modified software is installed. The new modules are coded with the help of structured design created in the design step. Every programmer is expected to do unit testing in parallel.
  5. **System Testing:** In this step, the particular software testing (integration testing) is performed on the modified system to ensure that no new faults are introduced in the software as a result of the maintenance activity. The input attribute comprises the updated software documentation, test preparation review report, and the updated system. Integration testing is done among newly created modules. Integration testing is also carried out between new modules and the

system. Finally the system is tested as a whole, following regressive testing procedures.

6. **Acceptance Testing:** In this step, acceptance testing is performed on the fully integrated system by the user or by a third party specified by the user. The objective is to detect errors and verify that the software features are according to the requirements stated in the modification request.
7. **Delivery:** After acceptance test, the system is deployed all over the organization either by small update package or fresh installation of the system. The final testing takes place at client end after the software is delivered. Training facility is provided if required, in addition to the hard copy of user manual. In this step, the modified (or new) software system is actually delivered to the user.
8. **Maintenance management:** Configuration management is an essential part of system maintenance. It is aided with version control tools to control versions, semi-version or patch management.

7.3 SOFTWARE REENGINEERING

- Reengineering is defined as, "the systematic transformation of an existing system into a new form to realize quality improvements in operation, system capability, functionality, performance, or evolvability at a lower cost, schedule or risk to the customer".

OR

- Reengineering is defined as, "the examination and alteration of software to reconstitute it in a new form, and includes the subsequent implementation of the new form."
- Software reengineering is a form of modernization that improves capabilities and/or maintainability of a system by introducing modern technologies and practices.
- Legacy software cannot keep tuning with the latest technology available in the market. Even if software grows old with time, its functionality does not. For example, initially Unix was developed in assembly language. When language C came into existence, Unix was reengineered in C, because working in assembly language was difficult.
- Other than this, sometimes programmers notice that few parts of software need more maintenance than others and they also need reengineering.

Activities of Reengineering:

- Fig. 7.5 shows Reengineering process. Reengineering Process has following activities:
  1. **Decide** what to reengineer. Is it whole software or a part of it?
  2. **Perform** reverse engineering, in order to obtain specifications of existing software.

3. **Re-structure** program if required. For example, changing function oriented programs into object-oriented programs.
4. **Re-structuring** data as required.
5. **Apply Forward Engineering** concepts in order to get reengineered software.

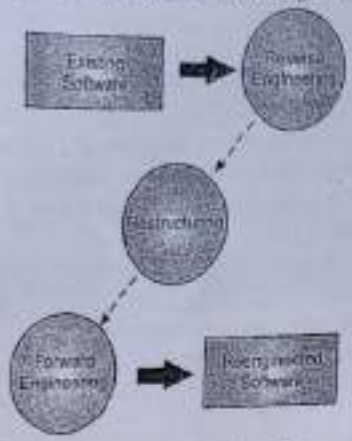


Fig. 7.5: Process of Reengineering

Advantages of Reengineering:

1. **Reengineering reduces the Risks:** There may be high risks if we try developing new software, risks may be related to staffing problems, design problems, specifications problems and so on.
2. **Reengineering improves Program Structure:** Refactoring is a reengineering technique that aims at reorganizing a program without changing its behavior. It seeks to improve a program structure and its maintainability.
3. **Reengineering reduces the Cost:** The cost of reengineering is usually lesser than the cost of developing new software. It also saves lot of time.

7.4 REVERSE ENGINEERING

- It is a process to achieve system specification by thoroughly analyzing, understanding the existing system.
- This process can be seen as reverse SDLC model, i.e. we try to get higher abstraction level by analyzing lower abstraction levels.
- An existing system is previously implemented design, about which we know nothing. Designers then do reverse engineering by looking at the code and try to get the design. With design in hand, they try to conclude the specifications going in reverse from code to system specification.

- The program itself is unchanged by the reverse engineering process.
- The software source code is usually available as the input to the reverse engineering process. Sometimes, however, even this has been lost and the reverse engineering must start with the executable code.
- Reverse engineering is not the same thing as reengineering. The objective of reverse engineering is to derive the design or specification of a system from its source code.
- Reverse engineering is used during the software reengineering process to recover the program design which engineers use to help them understand a program before re-organising its structure.
- Reverse engineering defined as, "the process of analyzing a subject system to identify the system's components and their interrelationships and create representations of the system in another form or at a higher level of abstraction".
- However, reverse engineering need not always be followed by reengineering.

Fig. 7.6: Reverse Engineering Process

- The term reengineering is also associated with Business Process Reengineering (BPR). BPR is concerned with redesigning business processes to reduce the number of redundant activities and improve process efficiency. It is usually reliant on the introduction or the enhancement of computer-based support for the process.

Advantages of Reverse Engineering:

1. Reverse engineering provides the abstract information from the detailed source code implementation.
2. Reverse engineering improves system documentation that is either incomplete or out of date.
3. It manages the complexity that is present in the software programs.
4. Reverse engineering focuses on recovering the lost information from the programs.

7.5 RESTRUCTURING AND FORWARD ENGINEERING

- In software engineering, restructuring is similar to reengineering. In re-structuring the transformation of a system from one representation to another happens at the same level of abstraction.

- In restructuring, functionality of the system does not change but only code cleaning happens. Revamping of the screens of the software can happen.
- Migration from one version of the software to another version of the software is the best example of re-structuring.
- Forward engineering starts with a system specification and involves the design and implementation of a new system.
- Fig. 7.7 shows forward engineering and reengineering concepts.

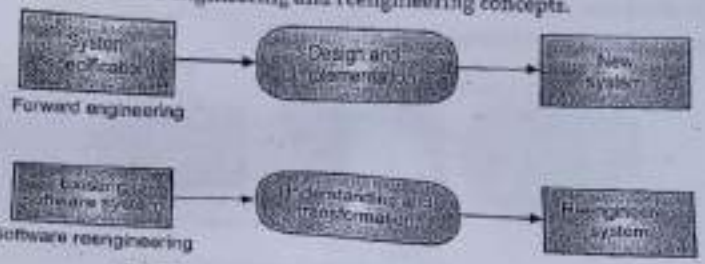


Fig. 7.7: Forward Engineering and Reengineering

- Software reengineering is any activity that improves one's understanding of software or prepares or improves the software itself, usually for increased maintainability, reusability, or evlovability.
- It is evident that reengineering entails some form of reverse engineering to create a more abstract view of a system, a re-generation of this abstract view followed by forward engineering activities to realize the system in the new form.

7.5.1 Restructuring

- Restructuring is defined as, "the process of transforming an existing system form one form to another without changing the logic or functionality".
- It is a process to restructure and re-construct the existing software. That means it is all about rearranging the source code, either in same programming language or from one programming language to a different one.
- Restructuring does not impact the functionality of the software but enhance reliability and maintainability.
- Program components, which cause errors very frequently can be changed, or updated with restructuring.
- The dependability of software on obsolete hardware platform can be removed via re-structuring.

- Software restructuring modifies source code and/or data in an effort to make it open to future changes. In general, re-structuring does not modify the overall program architecture. It tends to focus on the design details of individual modules and on local data structures defined within modules.

Process of Restructuring:

- Restructuring involves the following steps:

Step 1: Static analysis is performed, which provides information that is used to represent code as a directed graph or associative (semantic) network. The representation may or may not be in a human readable form; thus, an automated tool is used.

Step 2: Transformational techniques are used to refine (simplify) the representation.

Step 3: Refined representation is interpreted and used to generate the structured code.

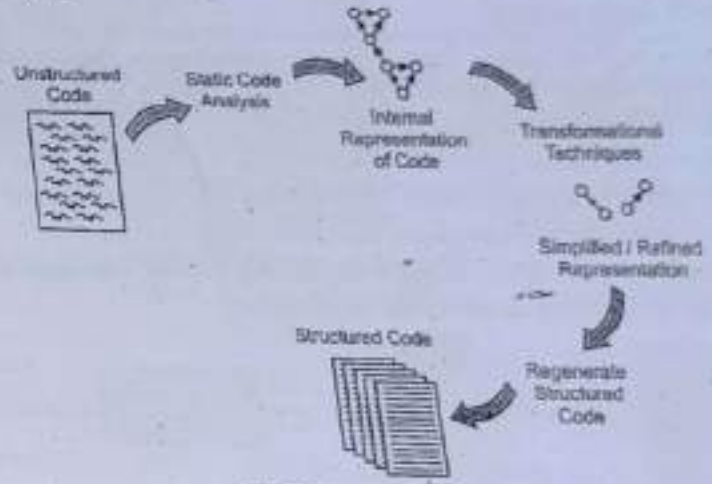


Fig. 7.8: Process of Restructring

- In software engineering, restructuring occurs when the basic architecture of application is solid, even though technical internals need work. It is initiated major parts of the software are serviceable and only a subset of all modules and need extensive modification.

Code Restructuring:

- It is performed to yield a design that produces the same function but with quality than the original program.

- In general, code restructuring techniques model program logic using Boolean algebra and then apply a series of transformation rules that produce restructured logic.

Data Restructuring:

- Data restructuring is a full scale engineering activity in which necessary data models are defined.
- Before data re-structuring can begin, a reverse engineering activity called analysis of source code should be conducted. All programming language statements that contain data definitions, file descriptions, I/O, and interface descriptions are evaluated.
- The intent is to extract data items and objects, to get information on data flow, and to understand the existing data structures that have been implemented. This activity is sometimes called Data Analysis.

7.5.2 Forward Engineering

- Forward engineering is defined as, "the traditional process of moving from high-level abstractions and logical, implementation-independent designs to the physical implementation of a system".
- Forward engineering is a process of obtaining desired software from the specifications in hand which were brought down by means of reverse engineering.
- It assumes that there was some software engineering already done in the past.
- Forward engineering is same as software engineering process with only one difference – it is carried out always after reverse engineering.

Fig. 7.9: Forward Engineering Process

Component Reusability:

- A component is a part of software program code, which executes an independent task in the system.
- It can be a small module or sub-system itself.
- **Example:** The login procedure used on the web can be considered as components, printing system in software can be seen as a component of the software.
- Components have high cohesion of functionality and lower rate of coupling, i.e. they work independently and can perform tasks without depending on other modules.

- In OOP, the objects are designed are very specific to their concern and have fewer chances to be used in some other software.
- In modular programming, the modules are coded to perform specific tasks which can be used across number of other software programs.
- There is a whole new vertical, which is based on re-use of software component, and is known as Component Based Software Engineering (CBSE).

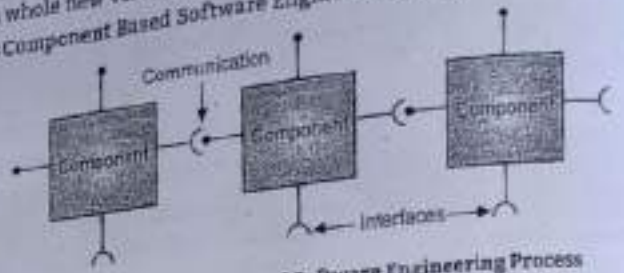


Fig. 7.10: Component Based Software Engineering Process

Levels of Re-use Process:

- Re-use can be done at follows levels:
  1. **Application level:** Where an entire application is used as sub-system of new software.
  2. **Component level:** Where sub-system of an application is used.
  3. **Modules level:** Where functional modules are re-used.
- Software components provide interfaces, which can be used to establish communication among different components.

Re-use Process:

- Two kinds of methods that can be adopted either by keeping requirements same and adjusting components or by keeping components same and modifying requirements.
- Fig. 7.11 shows following steps of Re-use Process:
  - **Step 1: System Requirements:** The functional and non-functional requirements are specified, which a software product must comply to, with the help of existing system, user input or both.
  - **Step 2: System Design:** This is also a standard SDLC process step, where requirements are defined in terms of software language. Basic architecture of system as a whole and its sub-systems are created.
  - **Step 3: Specify Components:** By studying the software design, the designers divide the entire system into smaller components or sub-systems. One complete

software design turns into a collection of a huge set of components working together.

Step 4: **Search Suitable Components:** The software component repository is referred by designers to search for the matching component, on the basis of functionality and intended software requirements.

Step 5: **Incorporate Components:** All matched components are packed together to shape them as complete software.

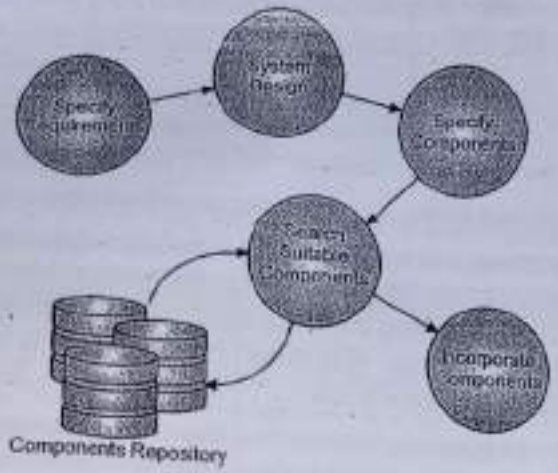


Fig. 7.11: Process of Re-use

7.5.4 Relationship between Forward, Reverse, Reengineering and Re-structuring

- In Fig. 7.12, the process of reengineering initiates with reverse engineering, an analysis of the original product that includes design recovery originating from the implementation phase and the design phase, restructuring the requirements of the system(data-to-data) and the design(graphical and functional).
- Once the new and modified requirements are defined, the second part of the reengineering process, the forward engineering continues the process with the development in order to achieve a new system.
- In the last step of reengineering, restructuring and re-documentation is applied. Re-documentation is the creation or revision of existing documentation, based on the original system.

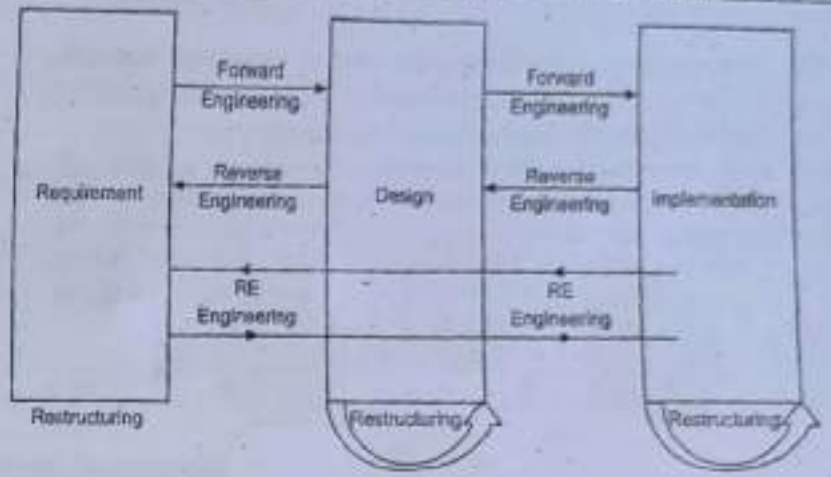


Fig. 7.12: Forward, Reverse, Re-engineering, Restructuring Concepts

Table 7.1: Difference between Forward and Reverse Engineering

	Forward Engineering	Reverse Engineering
Basic	Development of the application with provided requirements.	The requirements are deduced from the given application.
Purpose	Forward engineering employs the change in the subject system during restructuring.	Purpose is to examine the system to obtain its more abstract design.
Nature	Prescriptive nature	Adaptive
Needed skills	High proficiency	Low-level expertise
Accuracy	Model must be precise and complete.	Inexact model can also provide partial information.

Summary

- The software maintenance process comprises a set of software engineering activities that occur after the software has been delivered to the user.
- Software maintenance in software engineering is the modification of a software product after delivery to correct faults, to improve performance of the software system.
- Reengineering is an extension of reverse engineering. This technique refers to th systematic transformation of the present software system into a new form to mak

software design turns into a collection of a huge set of components working together.

Step 4: **Search Suitable Components:** The software **component repository** is referred by designers to search for the matching component, on the basis of functionality and intended software requirements.

Step 5: **Incorporate Components:** All matched components are packed together to shape them as complete software.

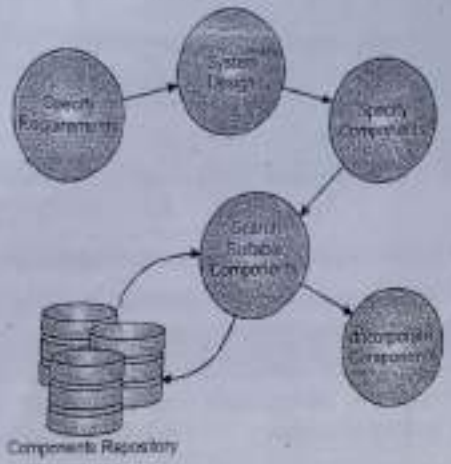


Fig. 7.11: Process of Re-use

7.5.4 Relationship between Forward, Reverse, Reengineering and Re-structuring

- In Fig. 7.12, the process of reengineering initiates with reverse engineering, an analysis of the original product that includes design recovery originating from the implementation phase and the design phase, restructuring the requirements of the system(data-to-data) and the design(graphical and functional).
- Once the new and modified requirements are defined, the second part of the reengineering process, the forward engineering continues the process with the development in order to achieve a new system.
- In the last step of reengineering, restructuring and re-documentation is applied. Re-documentation is the creation or revision of existing documentation, based on the original system.

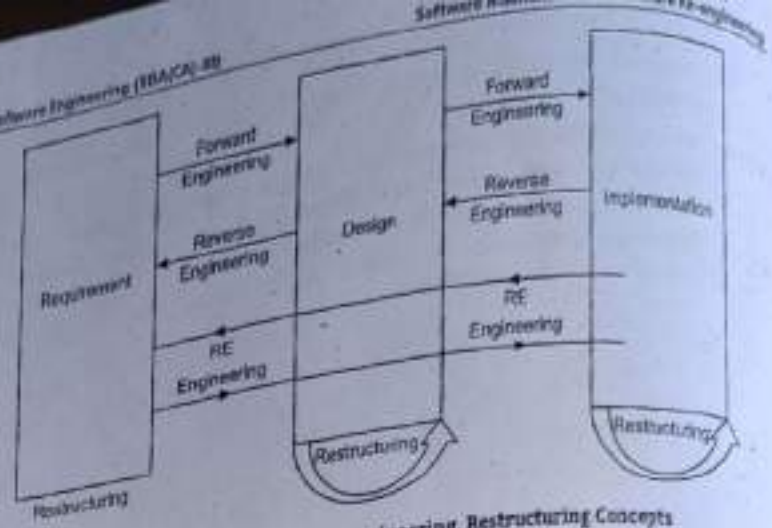


Fig. 7.12: Forward, Reverse, Re-engineering, Restructuring Concepts

Table 7.1: Difference between Forward and Reverse Engineering

	Forward Engineering	Reverse Engineering
Basic	Development of the application with provided requirements.	The requirements are deduced from the given application.
Purpose	Forward engineering employs the change in the subject system during restructuring.	Purpose is to examine the system to obtain its more abstract design.
Nature	Prescriptive nature	Adaptive
Needed skills	High proficiency	Low-level expertise
Accuracy	Model must be precise and complete.	Inexact model can also provide partial information.

Summary

- The software maintenance process comprises a set of software engineering activities that occur after the software has been delivered to the user.
- Software maintenance in software engineering is the modification of a software product after delivery to correct faults, to improve performance of the software system.
- Reengineering is an extension of reverse engineering. This technique refers to the systematic transformation of the present software system into a new form to make

quality improvements in operation, system capability, functionality, and achieving high performance at low costs.

- There are four types of maintenance namely, Corrective, Adaptive, Perfective, and Preventive.
- Corrective maintenance is concerned with fixing errors that are observed when the software is in use.
- Adaptive maintenance is concerned with the change in the software that takes place to make the software adaptable to new environment such as to run the software on a new operating system.
- Perfective maintenance is concerned with the change in the software that occurs while adding new functionalities in the software.
- Preventive maintenance involves implementing changes to prevent the occurrence of errors.
- When we need to update the software to keep it to the current market, without impacting its functionality, it is called Software Reengineering. It is a thorough process where the design of software is changed and programs are re-written.
- Reverse Engineering is a process to achieve system specification by thoroughly analyzing, understanding the existing system. This process can be seen as reverse SDLC model, i.e. we try to get higher abstraction level by analyzing lower abstraction levels.
- Forward engineering is a process of obtaining desired software from the specifications in hand which were brought down by means of reverse engineering.
- Restructuring involves the transformation of unstructured code into structured code thereby making it easier to understand and change.

Check Your Understanding

1. In how many categories software maintenance is being explained?
   (a) Two (b) Three
   (c) Four (d) Five
2. Which type of software maintenance is used for the modification of software that match changes in the ever – changing environment?
   (a) Adaptive (b) Perfective
   (c) Corrective (d) Preventive
3. In order to get re-engineered software, which concept is used?
   (a) Apply forward engineering (b) Perform
   (c) Restructure Program (d) None of the above
4. _____ extends the software beyond its original functional requirements.
   (a) Adaptive maintenance (b) Perfective maintenance
   (c) Corrective maintenance (d) Standard maintenance

5. This is the process of transfer mean and existing system from one form to another without changing the logic or functionality.
   (a) Reconstruction (b) Restructuring
   (c) Rebuild (d) Renew
6. _____ includes modifications and updations done in order to correct or fix problems, which are either discovered by user or conducted by user error reports.
   (a) Corrective maintenance (b) Adaptive maintenance
   (c) Perfective maintenance (d) Preventive maintenance

ANSWER KEY

1. (c)	2. (a)	3. (a)	4. (b)	5. (b)
6. (a)				

Practice Questions

Q.I: Answer the following Questions in short.

1. What is software maintenance?
2. Write different types of software maintenance methods.
3. What is forward engineering?
4. What is reverse engineering? Define it.
5. What is meant by restructuring?
6. What is code and data restructuring?

Q.II: Answer the following Questions.

1. Explain the term 'Restructuring' with its process.
2. Write short note on: Cost of maintenance.
3. Describe maintenance activities with the help of diagram.
4. Define "Reengineering". What are the advantages of Re-engineering?
5. With the help of diagram describe relationship between Forward, Reverse Reengineering and Re-structuring.
6. What is the need of software maintenance?

Q.III: Define the following terms.

1. Reengineering
2. Reuse process
3. Corrective Maintenance
4. Adaptive Maintenance
5. Preventive Maintenance
6. Code restructuring
7. Data restructuring

8... Agile Development

Objectives...

- To understand Basic Concepts of Agile Development
- To learn agile development methodologies such as XP, ASD and DSDM

8.1 INTRODUCTION

- The modern business environment has to respond to new opportunities and markets in a global world, economic conditions, and the emergence of competing products and services.
- New software is developed quickly to take advantage of new opportunities and to respond to competitive pressure.
- Agile software engineering represents a reasonable alternative to conventional software engineering to deliver successful software systems quickly.
- Agile software development is a group of software development methodologies based on iterative and incremental development, where requirements and solutions evolve through collaboration between self-organizing, cross-functional teams.
- Agile methods are processes that support the agile philosophy. Examples include Extreme Programming, Scrum, etc.
- Agile methods consist of individual elements called practices which include using version control, setting coding standards, and giving weekly demos to the stakeholders.

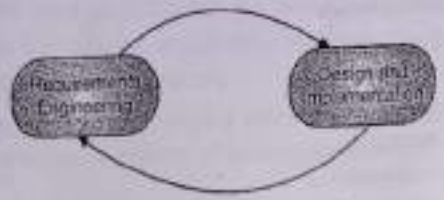


Fig. 8.1: Agile Development

- Agile is a widely popular software development method. Agile methods break the whole project into small incremental modules.
- Agile software development methodologies are an alternative waterfall to the traditional method of software development.

8.2 AGILITY

- Software development agility is the capability to manage various kinds of changes during the development process.
- Agility is dynamic, content specific, aggressively change embracing and growth oriented team able to appropriately respond to changes. According to Ivar Jacobson, "agility has become today's buzzword when describing a modern software process".

Creating and Responding to Change:

- Rapid changes in the software being built, changes to the team members, changes because of new technology, changes of all kinds that may have an impact on the product they build or the project that creates the product.
- Respond quickly and effectively to both anticipated and unanticipated changes.

Drawing the customer onto the team:

- Software engineers and other project stakeholders (managers, customers, and end users) work together on an agile team.
- An agile team in software process recognizes that software is developed by individuals working in teams and that the skills of these people, their ability to collaborate is at the core for the success of the project.
- An agile team in software process recognizes that software is developed by individuals working in teams and that the skills of these people, their ability to collaborate is at the core for the success of the project".
- Agility is more than an effective response to change:
  1. Agility emphasizes rapid delivery of operational software and it adopts the customer as a part of the development team and works on a project plan which must be flexible.
  2. Agility encourages team structures and attitudes that make communication (among team members, between technologists and business/organization people, between software engineers/developers and their managers) more facile.
- An agile development approach, emphasize an incremental delivery strategy that gets working software to the customer as rapidly as feasible for the product type and operational environment.
- Agile development is a different way of managing software development teams and projects. It is a set of activities that describes several agile methodologies. These methodologies included Scrum, XP (Extreme Programming), Crystal, and DSDM (Dynamic System Development Model).

8.3 AGILE PROCESS (BCA:W-16)

- Agile software process addresses following assumptions about the majority of software projects:
  - **Assumption 1:** It is difficult to predict in advance which software requirements will persist and which will change. It is equally difficult to predict how customer priorities will change as the project proceeds.
  - **Assumption 2:** For many types of software, design and construction are interleaved i.e., both activities should be performed in tandem so that design models are proven as they are created. It is difficult to predict how much design is necessary before construction is used to prove the design.
  - **Assumption 3:** Analysis, design, construction, and testing are not as predictable (from a planning point of view) as we might like.
- Given above assumptions, an important question arises: How do we create a process that can manage unpredictability? The answer lies in process adaptability. An agile software process must adapt incrementally. To accomplish incremental adaptation, an agile team requires customer feedback.

Advantages of Agile:

1. **Change is included:** With shorter planning cycles, it is easy and to accommodate and accept changes at any time during the software project.
2. **End-goal can be unknown:** Agile is very beneficial for projects where the end-goal is not clearly defined. As the project progresses, the goals will come to light and development can easily and simply adapt to these evolving requirements.
3. **Faster and High-quality Delivery:** Breaking down the project into iterations (manageable units) allows the software team to focus on high-quality development, with faster delivery.
4. **Strong Team Interaction:** Agile highlights the importance of frequent communication and face-to-face interactions. Teams work together and people are able to take responsibility and own parts of the projects.
5. **Continuous Improvement:** Agile projects encourage feedback from users and team members throughout the whole project, so lessons learned are used to improve future iterations.

Disadvantages of Agile:

1. **Planning can be Less Concrete:** Agile is based on time-boxed delivery and project managers are often reprioritizing tasks, it is possible that some items originally scheduled for delivery may not be complete in time.
2. **Team must be Knowledgeable:** Agile teams are usually small, so team members must be highly skilled in a variety of areas.

3. **Time Commitment from Developers:** Active involvement and collaboration is required throughout the Agile process, which is more time consuming than a traditional approach.
4. **Final Product can be very different:** Agile is so flexible, new iterations may be added based on evolving customer feedback, which can lead to a very different final deliverable.

8.3.1 Agile Principles

- Traditional software development methodologies are failing too often and yielding expected outputs. Then several prominent software developers gathered to create a new foundation for the software development approach. As a result, the Agile Manifesto was introduced in 2001, first outlining the basic concepts of agile development.The agile Manifesto is based on 12 principles.
- The principles of agile process are given below:
  - **Principle #1:** In this methodology, highest priority is to satisfy the customer through early and continuous delivery of valuable software.
  - **Principle #2:** Accept changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
  - **Principle #3:** Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
  - **Principle #4:** Developing team and developers must work together daily throughout the project.
  - **Principle #5:** Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
  - **Principle #6:** The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
  - **Principle #7:** Working software is the primary measure of progress.
  - **Principle #8:** Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
  - **Principle #9:** Continuous attention to technical excellence and good design enhances agility.
  - **Principle #10:** Simplicity--the art of maximizing the amount of work not done--is essential.
  - **Principle #11:** The best architectures, requirements, and designs emerge from self-organizing teams.
  - **Principle #12:** At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

8.3.2 Human Factors

(BCA: W-18, S-19)

- The scientist Cockburn and Highsmith said that the agile development focuses on the talents and skills of individuals and modeling the process to specific peoples and team.
- The process molds to the needs of people and team.

Characteristics of Agile development Team:

- The agile software project team must have the following characteristics for software development.
  1. **Competence:** Competence in the agile development context includes natural talent and specific software related skills with overall technical knowledge.
  2. **Collaboration:** Software development is about analyzing, assessing and using information that is communicated to the software project development team. Team members must collaborate with each other to have smooth development of software.
  3. **Common Focus:** The team members may perform various tasks and bring different skills to the project. All skills should be focus on a goal that is to deliver working software to the customer within a given time. To achieve this goal they have to focus on continual adaptations which will make the requirement to fit in a process.
  4. **Decision Making Ability:** The project team must be allowed the freedom to control its own destiny, so they can take individual decision on various issues.
  5. **Self-Organization:** To get a number of technical benefits, the collaboration should be improved and team confidence should be boosted.
     - (i) The agile project team organizes itself for the work to be done.
     - (ii) The agile project team organizes the process to best accommodate its local environment.
     - (iii) The agile project team organizes the work schedule to best accomplish delivery of the software increment.
  6. **Mutual Trust and Respect:** The agile software project team must become DeMarco called a "jelled" team shows respect and trust which is necessary to make them strongly connected.
  7. **The Fuzzy Problem-Solving Ability:** The Agile software project team may continuously deal with ambiguity and changes. In some situation, the agile software project team must accept the fact that the problem they solving today may not be the problem that needs and requirements to be solved tomorrow. But lessons learned from any problem solving activity may be of a benefit to the team later in the software project.

8.4 EXTREME PROGRAMMING

(BCA: S-18, W-19)

- Extreme Programming (XP) is originally described by Kent Beck, which is one of the most popular agile methodologies. Extreme Programming is a set of practices that conforms to the values and principles of Agile.
- XP is a discrete method, whereas Agile is a classification. There are many Agile methods. XP is just one of them. Kent Beck's basic idea was, 'Take observed effective team practices' and 'Push them to extreme level'.
- XP is a disciplined approach to delivering high-quality software quickly and continuously.
- It improves a software project in five essential ways i.e., communication, simplicity, feedback, respect, and courage.
- Extreme Programming promotes high customer involvement, rapid feedback loops, continuous testing, continuous planning, and close teamwork to deliver working software at very frequent intervals, typically every 1-3 weeks.
- Extreme Programming emphasizes teamwork. Managers, customers, and developers together make a team.

8.4.1 Practices in XP

- The original XP is based on four simple values: simplicity, communication, feedback, and courage with additional supporting practices as shown in Fig. 8.2.

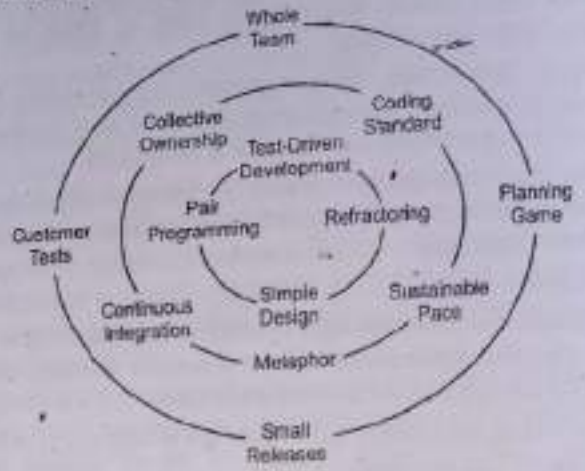


Fig. 8.2: Practices in XP

Various practices in Fig. 8.2 are explained below:

1. **Whole Team:** Within XP, the "customer" is not the one who pays the bill, but the one who really uses the system. XP says that the customer should be on hand at all times and available for questions. For instance, the team developing a financial administration system should include a financial administrator.
2. **Planning Game:** The main planning process within Extreme Programming is called the Planning Game. The game is a meeting that occurs once per iteration, typically once a week. The planning process is divided into two parts

   (i) **Release Planning:** The purpose of the Planning Game is to guide the product into delivery. This is focused on determining what requirements are included in which near-term releases, and when they should be delivered. The customers and developers are both parts of this.

   Release planning consists of three phases:

   (a) **Exploration Phase:** In this phase the customer will provide a shortlist of high-value requirements for the system. These will be written down on user story cards.

   (b) **Commitment Phase:** Within the commitment phase business and developers will commit themselves to the functionality that will be included and the date of the next release.

   (c) **Steering Phase:** In the steering phase the plan can be adjusted, new requirements can be added and/or existing requirements can be changed or removed.

   (ii) **Iteration Planning:** This plans the activities and tasks of the developers. In this process the customer is not involved. Iteration Planning also consists of three phases:

   (a) **Exploration Phase:** Within this phase the requirement will be translated to different tasks. The tasks are recorded on task cards.

   (b) **Commitment Phase:** The tasks will be assigned to the programmers and the time it takes to complete will be estimated.

   (c) **Steering Phase:** The tasks are performed and the end result is matched with the original user story.

3. **Small Releases:** The delivery of the software is done via frequent releases of live functionality creating concrete value. The small releases help the customer to gain confidence in the progress of the project. This helps maintain the concept of the whole team as the customer can now come up with his suggestions on the project based on real experience.
4. **Customer Tests:** The developers continually write unit tests, which need to pass for the development to continue. The customers write tests to verify that the features are implemented. The tests are automated so that they become a part of the system and

can be continuously run to ensure the working of the system. The result is a system that is capable of accepting change.

5. **Collective Ownership:** Collective code ownership (also known as "team code ownership" and "shared code") means that everyone is responsible for all the code; therefore, everybody is allowed to change any part of the code.
6. **Coding Standard:** Coding standard is an agreed upon set of rules that the entire development team agree to adhere to throughout the project. The standard specifies a consistent style and format for source code, within the chosen programming language, as well as various programming constructs and patterns that should be avoided in order to reduce the probability of defects.
7. **Sustainable Pace:** The concept is that programmers or software developers should not work more than 40 hour week and if there is overtime one week, that the next week should not include more overtime. Since the development cycles are short cycles of continuous integration, and full development (release) cycles are more frequent, the projects in XP do not follow the typical crunch time that other projects require (requiring overtime).
8. **Metaphor:** A metaphor is an expression, often found in the literature that describes a person or object by referring to something that is considered to have similar characteristics to that person or object.
9. **Continuous Integration:** Code is integrated and tested many times a day, one set of changes at a time. A simple way to do this is to have a machine dedicated to integration. The development team should always be working on the latest version of the software. Since different team members may have versions saved locally with various changes and improvements, they should try to upload their current version to the code repository every few hours, or when a significant break presents itself. Continuous integration will avoid delays later on in the project cycle, caused by integration problems.
10. **Test Driven Development:** Unit tests are automated tests that test the functionality of pieces of the code (e.g. classes, methods). Within XP, unit tests are written before the eventual code is coded. This approach is intended to stimulate the programmer to think about conditions in which his or her code could fail. XP says that the programmer is finished with a certain piece of code when he or she cannot come up with any further conditions under which the code may fail.
11. **Refactoring:** When implementing a feature, the developers always ask if there is a way of changing the existing code to make adding the feature simple. After they have added a feature, the developers ask if they now can see how to make the code simple while still running all of the tests. They restructure the system without changing behavior to remove duplication, improve communication, simplify, or add flexibil This is called refactoring.

12. **Simple Design:** Programmers should take a "simple is best" approach to software design. Whenever, a new piece of code is written, the author should ask themselves: Is there a simpler way to introduce the same functionality? If the answer is yes, the simpler course should be chosen. Refactoring should also be used, to make complex code simpler.

13. **Pair programming:** Pair programming means that all code is produced by two people programming on one task on one workstation. One programmer has control over the workstation and is thinking mostly about the coding in detail. The other programmer is more focused on the big picture, and is continually reviewing the code that is being produced by the first programmer. Programmers trade roles after a minute to hour periods. The pairs are not fixed; programmers switch partners frequently, so that everyone knows what everyone is doing, and everybody remains familiar with the whole system, even the parts outside their skill set. This way, pair programming also can enhance team-wide communication.

8.4.3 Process of XP

(BCA: W-18)

- XP is a lightweight, efficient, low-risk, flexible, predictable, scientific, and fun way to develop software.
- XP was conceived and developed to address the specific needs of software development by small teams in the face of vague and changing requirements.
- XP is one of the agile software development methodologies. It provides values and principles to guide team behavior. The team is expected to self-organize.
- Extreme programming encompasses a set of rules and practices that occur within the context of four framework activities: planning, design, coding, and testing.

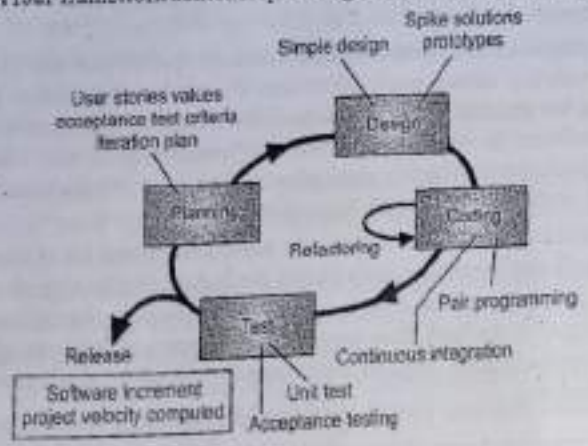


Fig. 8.3: Extreme Programming Process

- Fig. 8.3 shows the following phases of XP process:

1. **Planning:**
- The planning activity begins with the creation of a set of stories that describe required features and functionality for software to be built.
- Each story is written by the customer and is placed on an index card. The customer assigns a value to the story based on the overall business value of the feature of function.
- Members of the XP team then assess each story and assign a cost – measured in development weeks – to it.
- If the story will require more than three development weeks, the customer is asked to split the story into smaller stories and the assignment of value and cost occurs again.
- Customers and the XP team work together to decide how to group stories into the next release to be developed by the XP team.
- Once a basic commitment is made for a release, the XP team orders the stories that will be developed in one of three ways:
  (i) All stories will be implemented immediately.
  (ii) The stories with highest value will be moved up in the schedule and implemented first.
  (iii) The riskiest stories will be moved up in the schedule and implemented first.
- As development work proceeds, the customer can add stories, change the value of an existing story, split stories or eliminate them.
- The XP team then reconsiders all remaining releases and modifies its plan accordingly.

2. **Design:**
- XP design follows the Simple design principle. A simple design is always preferred over a more complex representation.
- The design provides implementation guidance for a story as it is written – nothing less, nothing more.

3. **Coding:**
- XP recommends that after stories are developed and preliminary design work is done, the team should not move to cord, but rather develop a series of unit test that will be exercise each story.
- Once the unit test has been created, the developer better able to focus on what must be implemented to pass the unit test.
- Once the code complete, it can be unit tested immediately, thereby providing instantaneous feedback to the developer.

- A key concept during the coding activity is pair programming. XP recommends that two people work together at one computer workstation to create code for a story. This provides a mechanism for real time problem solving and real time quality assurance.
- As pair programmers complete their work, the code they developed is integrated with the work of others.
- This continuous integration strategy helps to avoid compatibility and interfacing problems and provides a smoke testing environment that helps to uncover errors early.

4. **Testing:**

- The creation of unit test before coding is the key element of the XP approach.
- The unit tests that are created should be implemented using a framework that enables them to be automated. This encourages regression testing strategy whenever code is modified.
- Individual unit tests are organized into a "Universal Testing Suit", integration and validation testing of the system can occur on a daily basis. This provides the XP team with a continual indication of progress and also can raise warning flags early if things are going away.
- XP acceptance test, also called customer test, are specified by the customer and focus on the overall system feature and functionality that are visible and reviewable by the customer.

8.4.3 Advantages and disadvantages of Extreme Programming (XP)

Advantages of Extreme Programming (XP):

1. **Emphasis on Customer Involvement:** A major help to projects where it can be applied.
2. **Continuous Measurement:** Since software development is a people-intensive process, the principal measures concern people. It is therefore important to involve the programmers in measuring their own work.
3. **Incremental Development:** Consistent with most modern development methods.
4. **Simple Design:** Though obvious, worth stressing at every opportunity.
5. **Frequent Redesign (or Refactoring):** A good idea but could be troublesome with any but the smallest projects.
6. **Continuous Reviews:** A very important practice that can greatly improve any programming team's performance.
7. **Business Changes:** Changes are considered to be inevitable and are accommodated at any point of time.

8. **Production and Post-delivery Defects:** It is emphasis is on the unit tests to detect and fix the defects early.
9. **Timely Delivery:** Slipped schedules and achievable development cycles ensure timely deliveries.

Disadvantages of Extreme Programming (XP):

1. XP has lack of structure and necessary documentation.
2. It can be very inefficient, if the requirements for one area of code change through various iterations, the same programming may need to be done several times over.

8.5 ADAPTIVE SOFTWARE DEVELOPMENT (ASD) (BCA: W-18)

- Adaptive Software Development (ASD) is a technique for building complex software systems. ASD is a design principle for the creation of software systems. The principle focuses on the rapid creation and evolution of software systems.

ASD Life Cycle:

- Adaptive software development is made of three steps (See Fig. 8.4), each revolving around the coding of a program.
  - **Step 1: Speculation** : During this phase, coders try to understand the exact nature of the software and the requirements of the users. This phase relies on bugs and user reports to guide the project. If no reports are available, the developers use the basic requirements outlined by the purchaser.
  - **Step 2: Collaboration:** Collaboration in ASD is a balance between managing the doing and creating and maintaining the collaborative environment needed for output. As projects become increasingly complex, the balance swings much more toward the latter.
  - **Step 3: Learning :** Collaborative activities build products while Learning activities expose those products to a variety of stakeholders to ascertain value. Customer Focus Groups, Technical Reviews, Beta Testing, and Postmortems are all practices that expose results to scrutiny. In an adaptive environment, learning challenges all stakeholders, including both developers and customers, to examine their assumptions and use the results of each development cycle to learn the direction of the next.
- The ASD life cycles need to be short, so teams can learn from small rather than large mistakes. They also need to be double-loop, so teams learn both about product changes and more fundamental changes in underlying assumptions about how products are being developed.

- Fig. 8.4 shows the ASD life cycle.

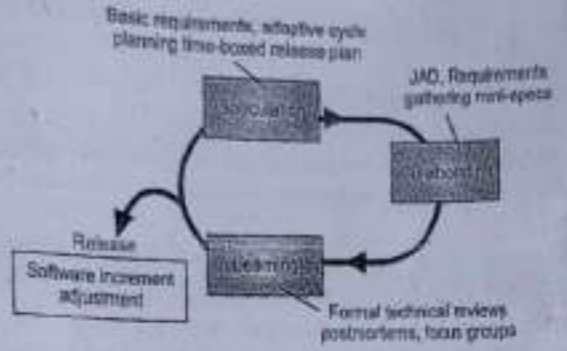


Fig. 8.4: ASD Life Cycle

8.6 SCRUM [RCA: S-19]

- Scrum is also an agile development method, which concentrates particularly on how to manage tasks within a team-based development environment.
- Scrum is the most popular and widely adopted agile method as it is relatively simple to implement and addresses many of the management issues.
- Scrum is an iterative and incremental agile software development framework for managing software projects and application development.
- Scrum focuses on 'a flexible, holistic product development strategy where a development team works as a unit to reach a common goal' as opposed to a 'traditional, sequential approach'.
- A 'process framework' is a particular set of practices that must be followed in order for a process to be consistent with the framework. (For example, the Scrum process framework requires the use of development cycles called Sprints, the XP framework requires pair programming, and so forth.) 'Lightweight' means that the overhead of the process is kept as small as possible, to maximize the amount of productive time available for getting useful work done.
- Scrum embraces uncertainty and creativity. Because that is how people work. It places a structure around the learning process, enabling teams to assess both what they have created, and just as importantly, how they created it.
- With Scrum methodology, the 'Product Owner' works closely with the team to identify and prioritize system functionality in form of a 'Product Backlog'.
- The Product Backlog consists of features, bug fixes, non-functional requirements, etc., whatever needs to be done in order to successfully deliver a working software system.

8.6.1 Activities of SCRUM

- The activities in Fig. 8.5 are explained below:

Sprint:

- Sprint is a pre-defined interval or the time frame (time box) in which the work has to be completed and make it ready for review or ready for production deployment.
- Sprints are time boxed so they always have a fixed start and end date, and generally they should all be of the same duration.

User Story:

- User stories are nothing but the requirements or feature which has to be implemented.
- In scrum, there are no huge requirements documents, rather the requirements are defined in a single paragraph, typically having the following format:

  As a <User / type of user>
  I want to <Some achievable goal / target>
  To achieve <some result or reason of doing the thing>

Story Points:

- Story points are quantitative indication of the complexity of a user story. Based on the story point, estimation and efforts for a story is determined.
- Story point is relative and is not absolute. To make sure that estimate and efforts are correct, it's important to check that the user stories are not big. Precise and smaller is the user story, accurate will be the estimation.

Scrum Roles:

- There are these core roles for producing the product:
  - (i) **Product Owner:** The Product Owner keeps track of the projects' stakeholders' expectations. It defines and gathers the required tools and resources that the Scrum Team needs. In addition, the Product Owner communicates its vision to the team to help set priorities. Scrum teams should have one, may also be a member of the development team.
  - (ii) **Scrum Master:** Scrum Master is responsible for removing weaknesses to the ability of the team to deliver the sprint goal/deliverables. This is not the team leader, but acts as a buffer between the team and any distracting influences. This ensures that the Scrum process is used as intended. This acts as enforcer of rules, Protector of the team and keep it focused on the tasks at hand. Differs from a Project Manager in that the latter may have people management responsibilities unrelated to the role of Scrum Master. Excludes any such additional people's responsibilities.

- Fig. 8.4 shows the ASD life cycle.

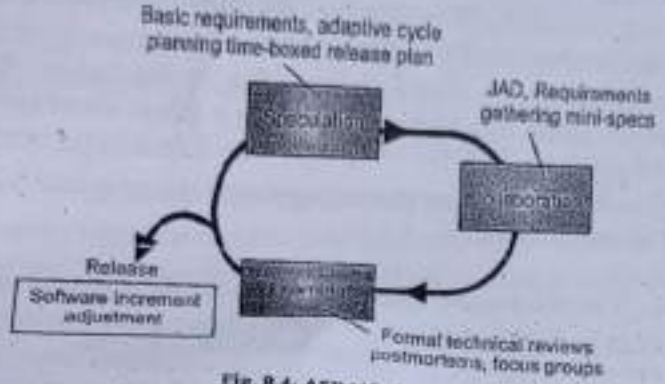


Fig. 8.4. ASD Life Cycle

8.6 SCRUM

(BCA: S-19)

- Scrum is also an agile development method, which concentrates particularly on how to manage tasks within a team-based development environment.
- Scrum is the most popular and widely adopted agile method as it is relatively simple to implement and addresses many of the management issues.
- Scrum is an iterative and incremental agile software development framework for managing software projects and application development.
- Scrum focuses on 'a flexible, holistic product development strategy where a development team works as a unit to reach a common goal' as opposed to a 'traditional, sequential approach'.
- A 'process framework' is a particular set of practices that must be followed in order for a process to be consistent with the framework. (For example, the Scrum process framework requires the use of development cycles called Sprints, the XP framework requires pair programming, and so forth.) 'Lightweight' means that the overhead of the process is kept as small as possible, to maximize the amount of productive time available for getting useful work done.
- Scrum embraces uncertainty and creativity. Because that is how people work. It places a structure around the learning process, enabling teams to assess both what they have created, and just as importantly, how they created it.
- With Scrum methodology, the 'Product Owner' works closely with the team to identify and prioritize system functionality in form of a 'Product Backlog'.
- The Product Backlog consists of features, bug fixes, non-functional requirements, etc., whatever needs to be done in order to successfully deliver a working software system.

8.6.1 Activities of SCRUM

- The activities in Fig. 8.5 are explained below:

Sprint:

- Sprint is a pre-defined interval or the time frame (time box) in which the work has to be completed and make it ready for review or ready for production deployment.
- Sprints are time boxed so they always have a fixed start and end date, and generally they should all be of the same duration.

User Story:

- User stories are nothing but the requirements or feature which has to be implemented.
- In scrum, there are no huge requirements documents, rather the requirements are defined in a single paragraph, typically having the following format:

  As a <User / type of user>
  I want to <Some achievable goal / target>
  To achieve <some result or reason of doing the thing>

Story Points:

- Story points are quantitative indication of the complexity of a user story. Based on the story point, estimation and efforts for a story is determined.
- Story point is relative and is not absolute. To make sure that estimate and efforts are correct, it's important to check that the user stories are not big. Precise and smaller is the user story, accurate will be the estimation.

Scrum Roles:

- There are these core roles for producing the product:
  - (i) **Product Owner:** The Product Owner keeps track of the projects' stakeholders' expectations. It defines and gathers the required tools and resources that the Scrum Team needs. In addition, the Product Owner communicates its vision to the team to help set priorities. Scrum teams should have one, may also be a member of the development team.
  - (ii) **Scrum Master:** Scrum Master is responsible for removing weaknesses to the ability of the team to deliver the sprint goal/deliverables. This is not the team leader, but acts as a buffer between the team and any distracting influences. This ensures that the Scrum process is used as intended. This acts as enforcer of rules, Protector of the team and keep it focused on the tasks at hand. Differs from a Project Manager in that the latter may have people management responsibilities unrelated to the role of Scrum Master. Excludes any such additional people's responsibilities.

(iii) **Development Team:** The Scrum Master and Product Owner organize and manage your Scrum Team that is actually doing the development. This team tends to be made up of several members that are cross-disciplinary, including engineers, designers, architects, and testers.

- The team is responsible for delivering potentially shippable product increments at the end of each Sprint. Made up of 3-9 people with cross-functional skills who do the actual work (analyze, design, develop, test, technical communication, document, etc.). This team is Self-organining. Eventhough they may interface with Project Management Organizations (PMOs).

8.6.2 SCRUM Artifacts

- Scrum's artifacts represent work or value to provide transparency and opportunities for inspection and adaptation. Artifacts defined by Scrum are specifically designed to maximize transparency of key information so that everybody has the same understanding of the artifact.

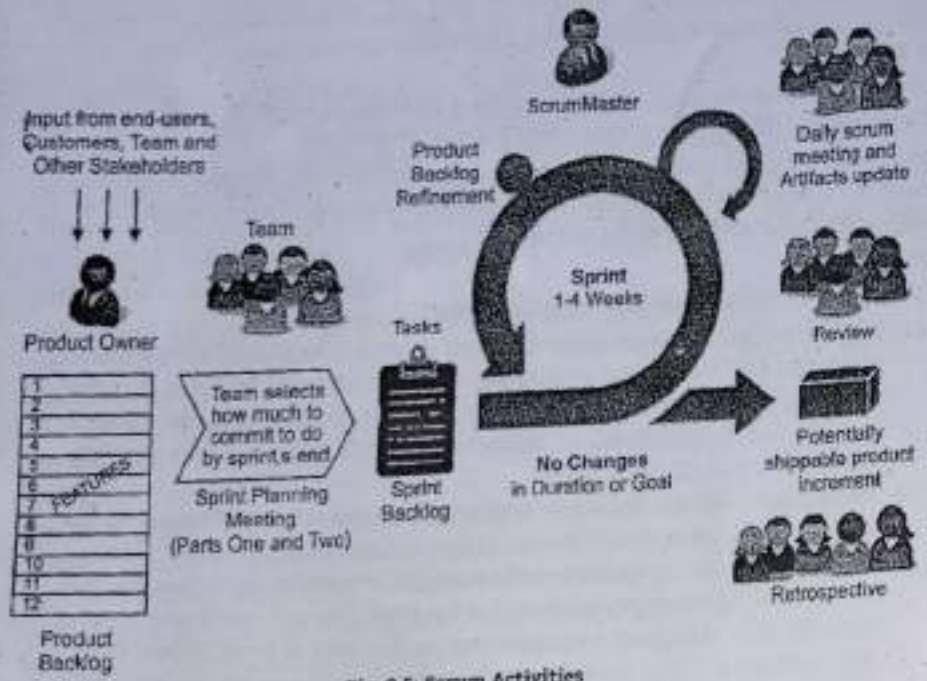


Fig. 8.5: Scrum Activities

Product Backlog:

- It is an ordered list of "requirements" that is maintained for a product. It consists of features, bug fixes, non-functional requirements, etc., whatever needs to be done in order to successfully deliver a working software system.
- In Scrum, it is not required to start a project with a lengthy, upfront effort to document all requirements. This agile product backlog is almost always more than enough for a first sprint.
- The Scrum product backlog is further allowed to grow and change as more is learned about the product and its customers.

Sprint Backlog:

- It is the list of work the Development Team must address during the next sprint.
- The list is derived by selecting stories/features from the top of the product backlog until the Development Team feels it has enough work to fill the sprint.

8.6.3 Processes in Scrum

- **Sprint Planning:** A sprint begins with the team importing stories from the release backlog into the sprint backlog; it is hosted by scrum master. The Testers estimate effort to test the various stories in the Sprint Backlog.
- **Daily Scrum:** It is hosted by scrum master, it takes about 15 minutes. During Daily Scrum, the members will discuss the work completed the previous day, the planned work for the next day and issues faced during a sprint. During daily stand-up meeting team progress is tracked.
- **Sprint Review/ Retrospective:** It is also hosted by scrum master, it last about 2-4 hours and discuss what the team has accomplished in the last sprint and what lessons were learned.

8.6.4 The Scrum Framework

- The product owner creates a product backlog (essentially, a wish list of tasks that need to be prioritized in a project).
- The Scrum team conducts a sprint planning session where the tasks necessary to complete items on the wish list is broken down into small, more easily manageable chunks.
- The team creates a sprint backlog and plans its implementation.
- The team decides time duration for every sprint (the most common intervals are probably two weeks).
- The team gets together every day for a brief Scrum meeting (often referred to as a Daily Standup) where each member of the team shares daily updates, helping the team and the project manager assess the progress of the project.

- The certified Scrum master guides the team and keeps them focused and motivated.
- The stakeholders and the product owner conduct a review at the end of each sprint.
- This is the cycle followed by a Scrum team in a product development project.

8.6.5 Advantages and Disadvantages of Scrum

Advantages of Scrum Software Development:

1. Scrum can help teams complete project deliverables quickly and efficiently.
2. It provides continuous feedback. This helps to make the project better in the long run.
3. A daily meeting easily helps the developer to make it possible to measure individual productivity.
4. In this methodology, large projects are divided into easily manageable sprints.

Disadvantages of Scrum Software Development:

1. This kind of development model is suffered if the estimating project costs and time will not be accurate.
2. It is good for small, fast moving projects but not suitable for large size projects.
3. This methodology needs experienced team members only. If the team consists of people who are novices, the project cannot be completed within exact time frame.

8.7 DYNAMIC SYSTEM DEVELOPMENT MODEL (DSDM)

- DSDM is dynamic in nature as it's a Rapid Application Development (RAD) approach to software development. It is an iterative and incremental approach that emphasizes continuous customer/ Client involvement.
- The DSDM methodology has evolved and matured to provide a comprehensive foundation for planning, managing, executing, and scaling agile process and iterative software development projects.
- The DSDM project framework is independent of, and can be implemented in conjunction with, other iterative methodologies such as Extreme Programming (XP) and the Rational Unified Process (RUP).

Features of DSDM:

- The main **features of the DSDM** method are as follows:
  1. User involvement.
  2. Iterative and incremental development.
  3. Increased delivery frequency.
  4. Integrated tests at each phase.
  5. The acceptance of delivered products depends directly on fulfilling requirements.

8.7.1 Principles of DSDM

- DSDM has eight principles supporting the framework. Each principle must be followed by the project team. Compromising any principle undermines the basic philosophy and introduces risk to the successful outcome of the project.
- The eight Principles are:
  1. Focus on the business need
  2. Deliver on time
  3. Collaborate
  4. Never compromise quality
  5. Build incrementally from firm foundations
  6. Develop iteratively
  7. Communicate continuously and clearly
  8. Demonstrate control

8.7.2 Tools and Techniques

- The main focus of DSDM is that the product is delivered frequently in each iteration. Different tools, techniques, and, practices are used to support the whole DSDM process.
- Most common and used practices from the DSDM framework which are:
  - Time Boxing
  - MoSCoW Rules
  - Facilitated workshop

1. **Time Boxing:** Time boxing is the amount of work to be done in a fixed given time. One software development project will have multiple time boxes, and every time box is assigned one specific objective. Time boxing helps in achieving the target in time.
2. **MoSCoW Rules:** MoSCoW is a simple prioritizing technique. This helps in understanding and prioritizing the tasks to be performed the letters here stand for:
   - **Must-Have:** The requirements that are fundamental and must conform to the solution.
   - **Should Have:** The things that are important for the business solution.
   - **Could Have:** The requirements that are important but can more easily be left out for a short while.
   - **Won't Have:** The requirements that can wait and include in later development.
3. **Facilitated workshop** : Facilitated Workshops means providing a workshop whe team members and developers can work on a clear pre-set of deliverables. To access to this workshop, they coordinate with a neutral person who is ca Workshop facilitator. The facilitator will be guiding the team through a proce achieve its objectives, expertly.

8.7.3 DSDM Software Development Process

- The method provides a four-phase framework consisting of:
  1. Feasibility and business study
  2. Functional model / prototype iteration
  3. Design and build iteration
  4. Implementation
- Fig. 8.6 shows the model of the DSDM software development process.

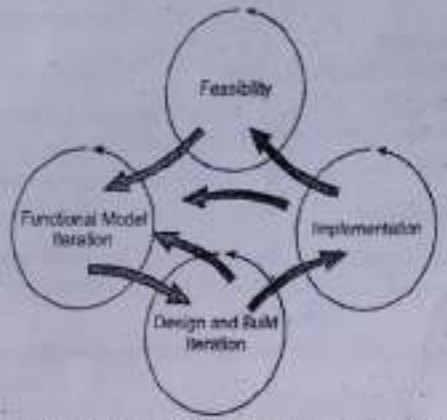


Fig. 8.6: Model of the DSDM software development process

- The phases of DSDM shown in above fig. are:

1. **Feasibility Study and Business Study:**

Feasibility Study:

- It establishes the basic business requirements and constraints associated with the application to be built and then assesses whether the application is a viable candidate for the DSDM process.
- It is about whether the proposed method can be applied or not. Through research it is carried out to find out the existing problems.

Business Study:

- It establishes the functional and information requirements that will allow the application to provide business value.
- It also defines the basic application architecture and identifies the maintainability requirements for the application.

- It is about acquiring a clear understanding of the business flow and how the processes are related to each other. It involves identifying the stakeholders and those who are involved in the project.

2. **Functional Model/ Prototype Iteration:**

- Functional model iteration produces a set of incremental prototypes that demonstrate functionality for the customer.
- It works on refining high level business information requirements and functions of systems identified during the business study of the methodology.
- In this phase, risk has to be identified and recognize a plan on how to deal with risk for future developments. The outcome of the functional model iteration is standard analysis model of the software.

3. **Design and Build Iteration:** In this phase, revisits prototypes built during functional model iteration to ensure that each has been engineered in a manner that will enable it to provide operational business value for end users.

- In this phase the actual system is built based on the non-functional requirements carried out in the previous phase and the built-in system is implemented in the next phase once the testing is done.

4. **Implementation:**

- This is the final phase in the methodology where the built-in system is moved into the production environment from the developed environment.
- Implementation phase places the latest software increment (an "operationalized" prototype) into the operational environment.

8.7.4 Advantages and disadvantages of DSDM

Advantages of the DSDM:

1. DSDM provides a technique-independent process and flexible in terms of requirement evolution.
2. DSDM designed from the grounded up by business people, so the business value is identified an expected to be the highest priority deliverable.
3. DSDM incorporates stakeholders into the development process.
4. An emphasis as DSDM on testing is so strong that at least one tester is expected to be on each project team. This improves software quality and performance.

Disadvantages of the DSDM:

1. DSDM focuses on RAD, which can lead to decrease in code robustness.
2. It requires significant user involvement.
3. DSDM requires a skilled development team in both the business and technical areas.

Summary

- Agile software development describes a set of values and principles for software development under which requirements and solutions evolve through the collaborative effort of self-organizing cross-functional teams.
- Agile software development advocates adaptive planning, evolutionary development, early delivery, and continuous improvement, and it encourages rapid and flexible response to change.
- The term agile was popularized by the Agile Manifesto, which defines those values and principles.
- Agile Development is an umbrella term for several iterative and incremental software development methodologies. The most popular agile methodologies include Extreme Programming (XP), Scrum, Dynamic Systems Development Method (DSDM) and so on.
- Agility has become today's buzzword when describing a modern software process. Everyone is agile. An agile team is a nimble team able to appropriately respond to changes. Change is what software development is very much about.
- In software application development, Agile is a methodology that anticipates the need for flexibility and applies a level of pragmatism into the delivery of the finished product.
- Agile requires a cultural shift in many companies because it focuses on the clean delivery of individual pieces or parts of the software and not on the entire application.
- Like all software technology arguments, agile development methodology debate risks degenerating into a religious war. If warfare breaks out, rational thought disappears and beliefs rather than facts guide decision-making.
- Agile development focuses on the talents and skills of individuals, molding the process to specific and teams. The key point in this statement is that the process molds to the needs of the people and team, not the other way around.
- Extreme Programming (XP) is a software development methodology which is intended to improve software quality and responsiveness to changing customer requirements. As a type of agile software development, it advocates frequent "releases" in short development cycles, which is intended to improve productivity and introduce checkpoints where new customer requirements can be adopted.
- XP encompasses a set of rules and practices that occur within the context of four framework activities: planning, design, coding and testing.
- XP is a set of practices that conforms to the values and principles of Agile. XP is a discrete method, whereas Agile is a classification.
- DSDM is a Rapid Application Development (RAD) approach to software development and provides an agile project delivery framework. The important aspect of DSDM is

that the users are required to be involved actively, and the teams are given the power to make decisions. Frequent delivery of products becomes an active focus with DSDM.

- Scrum is an iterative and incremental agile software development framework for managing software projects and product or application development. Its focus is on "a flexible, holistic product development strategy where a development team works as a unit to reach a common goal" as opposed to a "traditional, sequential approach".
- Backlog is a prioritized list of projects requirement or features that provide business value for the customers. Items can be added to the backlog at any time.
- The product backlog is an ordered list of "requirements" that is maintained for a product.
- The sprint backlog is the list of work the Development Team must address during the next sprint.
- In Scrum team, The Product Owner represents the stakeholders and is the voice of the customer.
- An ASD life cycle includes three phases i.e., Speculation, Collaboration, And Learning.
- The DSDM is an agile software development approach that provides a framework for building and maintaining systems, which meet time constraints through the use of incremental prototyping in a controlled project environment.

Check Your Understanding

1. What are the three framework activities of ASD process model?
   (a) Analysis, design, coding (b) Analysis, implementation, design
   (c) Initial study, planning, development (d) Speculation, collaboration, learning
2. Which is the phase of XP process?
   (a) Planning (b) Design
   (c) Coding (d) All of the above
3. Agile modeling provides guidance to Practioner during which of the following software tasks?
   (a) Analysis (b) Coding
   (c) Design (d) Both a and c
4. Which is not activity of scrum method?
   (a) Sprint (b) User story
   (c) Product master (d) Story points
5. ASD stands for_____.
   (a) Adaptive System Development (b) Analytical Software Development
   (c) Analog System Development (d) Adaptive Software Development

ANSWER KEY

1. (d)	2. (d)	3. (d)	4. (c)	5. (d)

Practice Questions

Q.I: Answer the following Questions in short.

1. What is the Agile process?
2. What is ASD?
3. What is Scrum in Agile development?
4. Define Extreme Programming (XP)?
5. Which human factors involved in agile process?
6. Which phases are including in ASD Life cycle?
7. Which are popular agile methodologies?

Q.II: Answer the following Questions.

1. Which are principles of Agile?
2. Explain in detail Phases of DSDM.
3. Write advantages and disadvantages of Agile Process.
4. Explain with neat diagram describe Scrum process.
5. Describe XP process diagrammatically.
6. Explain ASD life cycle.
7. What are advantages and disadvantages of DSDM?

Q.III: Define the following terms.

1. Agility
2. Extreme programming
3. Scrum Master
4. Product Owner
5. Sprint
6. Product Backlog
7. Story Points in Scrum
8. Refactoring
9. Speculation in ASD
10. Scrum Roles

Previous Exams Questions

BCA (Science)

Summer 2018

1. What is Agility? 1M
Ans. Refer to section 8.2
2. Explain any three human factors for agile process. 3M
Ans. Refer to section 8.3.3
3. Write a short note on Extreme programming values. 3M
Ans. Refer to section 8.4
4. Explain any (three) principal to achieve agility. 3M
Ans. Refer to section 8.3.1

Winter 2018

1. DSDM stands for ___ 1M
(a) Dynamic System Development Model
(b) Dynamic Software Deployment Model
(c) Dynamic System Deployment Model
(d) Dynamic Software Development Model
Ans. Refer to section 8.7
2. List any two phases of Adaptive Software Development life cycle. 1M
Ans. Refer to section 8.5
3. Explain any three human factors used for Agile process. 3M
Ans. Refer to section 8.3.3
4. State and explain any five principles of Agile process. 5M
Ans. Refer to section 8.3.1
5. Explain any two phases of "XP Process" along with a neat diagram. 3M
Ans. Refer to section 8.4.2

Summer 2019

1. What is scrum? 3M
Ans. Refer to section 8.5

◆◆◆

Bibliography

1. Software Engineering: A Practitioner's Approach: By Roger S. Pressman.
2. Agile Software Development: By Alistair Cockburn.
3. Software Engineering: By Subramanian Chandramouli, Saikat Dutt, Chandramouli Seetharaman, B. G. Geetha.
4. System Engineering Analysis, Design, and Development: Concepts, Principles....: By Charles S. Wasson.
5. Software Engineering : By Dr. (Prof.) Rajendra Prasad, Prof. Govind Verma.
6. Software Engineering : By Ian Somerville.
7. https://en.m.wikipedia.org/wiki/Software_testing.
8. https://www.tutorialspoint.com/software_testing.
9. https://en.wikipedia.org/wiki/Software_requirements.
10. https://www.workfront.com/project-management/methodologies/agile.

❖❖❖