UNIT - III

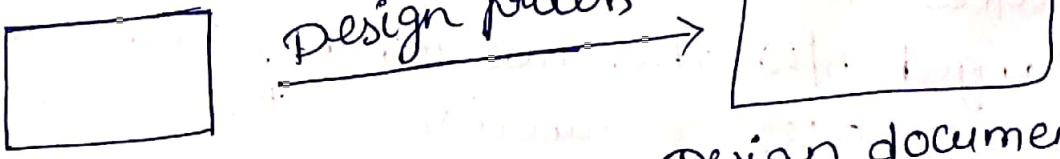
S/w Design (Function oriented & object oriented)

⇒ During the s/w design phase, the design document is produced, based on the customer Requirements as documented in the 'SRS' document.

(or)

The activities carried out during the design phase transform the SRS document into the design document.

The design process starts using the SRS document & completes with the production of the design document. the design document produced at the end of the design phase, should be implementing using a programming language in coding phase.



SRS document

Design document

Overview of the design process:-

The design process Essentially transforms the SRS document into a design document. the s/w design must be "Quality"

Outcome of the design process :

The following items are designed and documented during the design phase.

(1) Different Modules Required :

The diff" modules in the solution should be clearly identified. Each module is a collection of functions and the data shared by the functions of the module.

(2) control Relationship among modules :

The interfaces b/w A control Relationship b/w two modules essentially arises due to function calls across the 2' modules. Here various modules should be identified in the design document

(3) Interfaces among diff modules :

The interfaces b/w 2' modules identifies the exact data items that are exchanged b/w the two modules when one module invokes a function of the other module.

(4) Data Structures of the individual modules

Each module normally stores some data that the functions of the module need to share the overall responsibility of the module. a module is a part of a program.

(5) Algorithms Required to implement the Individual Modules:

Each function in a module usually performs some processing activity. the Algorithms required to processing activities of various modules need to be carefully designed & documented with due considerations given to the accuracy of the Results, space & time Complexities.

Starting with the SRS document the design documents are produced through iterations over a series of steps that we are going to discuss. S/W is divided into separately named & Addressable Components Called "Module"

Classification of Design Activities:

A good s/w design is seldom (rarely) realised by using a single step procedure, rather it requires iterating over a series of steps called the design activities.

Depending on the order in which various design activities are performed, we can broadly classify them into 2 imp" stages

(i) High-level design
(ii) Detailed design
(iii) Architectural Design → (People; Server; machines)

The architectural design is the highest abstract versio of the system; it identifies the s/w, as a system with many components interacting with Eachother

(i) High-level design :- (objected design)

In High-level design, a problem is decomposed into a set of modules. the control relationship among these modules are identified, and also the interfaces among various modules are identified.

The outcome of high-level design is called the program structure.

High-level design is a crucial step in the overall design of a s/w. when the high-level design is complete, the problem should have been decomposed into many small functionally independent modules that are cohesive, have low coupling among themselves and are arranged in a hierarchy. in Architectural design the designers get the idea of proposed solution domain.

(ii) Detailed design :- (Algorithms → (Pseudocode))

During the detailed design each module is examined carefully to design its data structures and the algorithms.

The outcome of detailed design is usually documented in the form of a module specification (MPSMSPEC) document.

Classification of Design Methodologies:

The design activities vary considerably based on the specific design methodology being used. A large no. of S/W design methodologies are available. We can roughly classify these methodologies into procedural & object-oriented approaches. These 2 approaches are 2 fundamentally diff. design paradigms.

Do Design Techniques Result in unique solution?

Even while using the same design methodology, diff. designers usually arrive at very diff. design solutions. it is possible that even the same designer can work out many diff. solutions to the same problem. Therefore, obtaining a good design would involve trying out several alternatives and picking out the best one.

Analysis Vs Design:-

The goal of any Analysis technique is to Elaborate the Customer Requirements

through careful thinking and at the same time consciously avoiding making any decisions regarding the exact way the system is to be implemented

The Analysis model is usually documented using some graphical formalism. The Analysis model would be documented using "Dataflow diagrams (DFD's)

The object-oriented approach, both the design model & analysis model will be documented using (UML). the Analysis model would normally be very difficult to implement using a programming language

How to Characterise a good S/W design

(i) correctness :-

A good design should first of all be correct. (i.e) it should correctly implement all the functionalities of system

(ii) understandability :- A good design should be easily understandable.

(iii) Efficiency :- A good design solution should adequately address Resource, time and

cost optimisation issues. Efficient & good Quality s/w developed.

(iv) Maintainability :-

A good design should be easy to change. since change Requests usually keep coming from the customer even after product Release. (v) Simplicity vi) completeness vii) verifiability viii) Portability ix) Reliability

understandability of a Design :- (A Major Concern)

Given that we are choosing from only correct design solutions, understandability of a design solution is possibly the most imp. issue to be considered while judging the goodness of a design.

An understandable design is modular and layered :

How can the understandability of 2 diff. designs be compared, so that we can pick the better one?

→ It should assign consistent and meaningful names to various design components

→ It should make use of the principles of decomposition & abstraction in good measures to simplify the design

Modularity :-

A modular design is an Effective decomposition of a problem. A design solution is said to be highly modular, if the diff" modules in the solution have high Cohesion & their inter-module Couplings are low. :- S/W sys" into multiple independent modules where each module works independently.

Layered design:

A layered design is one in which when the call relations among diff" modules are represented graphically. it would result in a tree-like diagram with clear layering. A good S/W design must be "Low module Coupling, High module Cohesion".

Cohesion & Coupling:

Cohesion:- These are the most popular Criterions used to measure the modularity in a system. An Effective modular system has "low Coupling & high Cohesion". "It is a strength of relation within modules".

Coupling:- It is an indication of the Relative interdependence among modules. ~~The strength of the Relationship b/w modules.~~

Functional Independence: (when a module focuses on a single task)

A module that is highly cohesive & also has low coupling with other modules is said to be functionally independent of the other modules. FI → should be able to accomplish it with very (Little interaction with other modules)

Advantages:

(i) Error Isolation:

whenever an error exists in a module, functional independence reduces the chances of the error propagating to the other modules.

(ii) Scope of Reuse:

Reuse of a module for the development of other applications becomes easier.

(iii) understandability:

when modules are functionally independent, complexity of the design is greatly Reduced. modules are less interaction (with other so can be understand in isolation

layered Arrangement of modules:

An important characteristic feature of a good design solution is layering of the modules. A layered design achieves

Control abstraction & is easier to understand and debug.

Imp.. Concepts in layered design:-

(i) Superordinate & Subordinate module:-

In a control hierarchy, a module that controls another module is said to be superordinate to it.

→ A module controlled by another module is said to be subordinate to the controller.

(ii) visibility:-

A module 'B' is said to be visible to another module A, if 'A' directly calls 'B'.

(iii) Control Abstraction:-

In a layered design, a module should only invoke the functions of the modules that are in the layer immediately below it

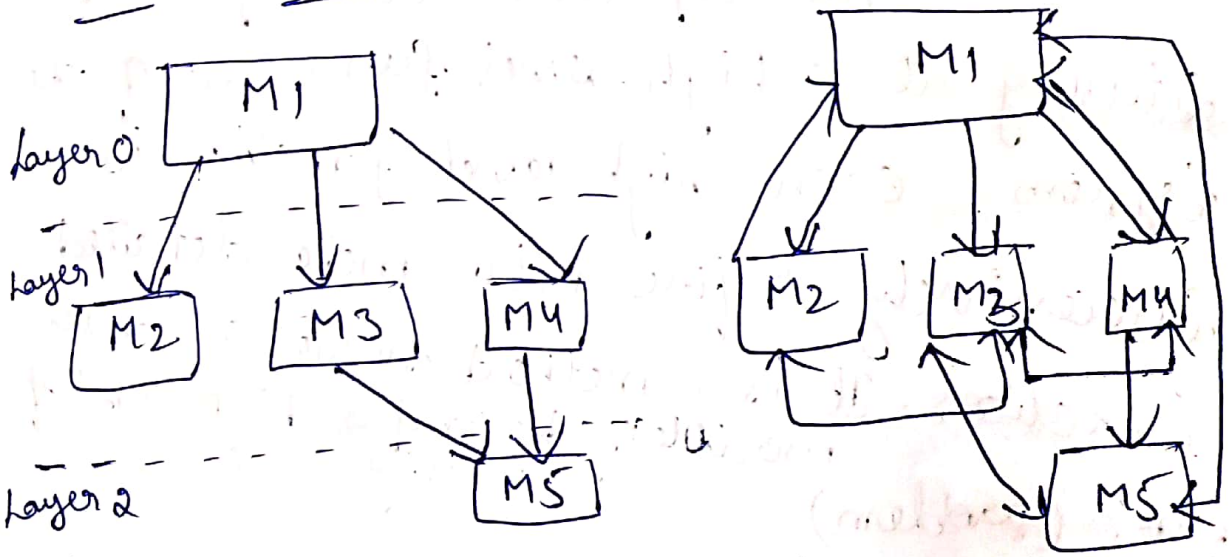
(iv) Depth & width:- this control hierarchy provide an indication of the no.. of layers & the overall span of control respectively.

v) Fan-out:-

It is a measure of the no.. of modules that are directly controlled by a given module.

vi) Fan-in:- It indicates the no.. of modules that directly invoke a given module.

Good & poor control Abstraction:-



a) Layered design with good control abstraction.

b) Layered design showing poor control Abstraction.

Approaches to S/w Design:-

(i) Function-oriented design

(ii) Object-oriented design

Function-Oriented Design:

features:

(i) Top-down decomposition:

A system, to start with, is viewed as a black box that provides a certain services to the users of the system.

The top-down decomposition, starting at a high-level fun view of the system, each high-level function is successively refine into more detailed functions. It is a method to s/w where the module is decomposed into a set of interacting units.

Ex:- (problem)

Consider a function "create-new-library member which essentially creates the Record for a new member, assigns a unique membership no" to him, and prints a bill towards his membership charge.

Sol:-

This high-level function may be refined into the following subfunctions.

(i) Assign-membership-no.

(ii) Create-member-Record

(iii) print-bill

Each of these subfunctions may be split into more detailed subfunctions.

Centralised system state:

The system state is centralised and shared among diff" functions.

Ex:-

In library management system, several functions such as the following share data such as member-records for Reference & updation:

Sol:- (i) create-new-member

(ii) Delete-member

(iii) Update-member-record

A large no" of function-oriented design approaches have been proposed in the past.

(ii) Object - Oriented design:

In OOD a system is viewed as being made up of a collection of objects (i.e Entities). Each object is associated with a set of functions that are called its methods. Each object contains its own data and is responsible for managing it. the data internal to an object can't be accessed directly by other objects and only through invocation of the methods of the objects. the system state is decentralised since there is no globally shared data in the system and data is stored in each object.

Ex:- In a library automation s/w, each library member may be a separate object with its own data & functions to operate on the stored data. the methods defined for one object can't directly refer to (or) change the data of other objects.

Difference b/w object-oriented and function oriented design:

Ex:

An Automated fire-alarm system for a large building. (Customer Requirements)

→ The owner of a large multi-storied building wants to have a computerised fire alarm system designed, developed, and installed in his building. Smoke detectors & fire alarms would be placed in each room of the building. the fire alarm system would ~~be placed in each~~ to monitor the status of these smoke detectors. whenever a fire condition is reported by any of the smoke detectors, the fire alarm system should determine the location at which the fire has been sensed & then sound the alarms only in the neighbouring locations

The fire alarm system should also flash an alarm message on the computer console. fire fighting personnel would man the console round the clock. After a fire condition has been successfully handled, the fire alarm system should support resetting the alarms by the fire fighting personel.

→ In function-oriented program, the system state (Data) is centralised and several functions access & modify this central data.

→ In object-oriented, the state info (data) is distributed among various objects.

→ In the oject-oriented design, data is private in diff objects & these are not available to the other objects for direct access & modification.

Function-oriented S/w design:

The term top-down decomposition is often used to denote the successive decomposition of a set of high-level functions into more detailed functions.

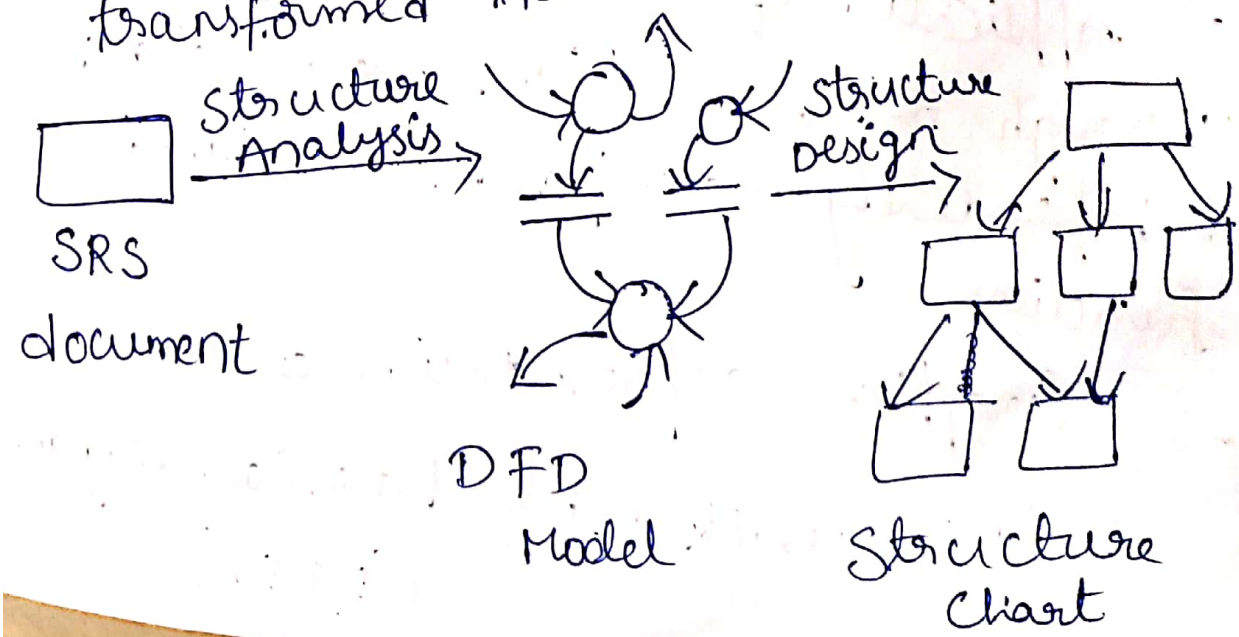
Overview of SA/SD Methodology

(Structure Analysis / Structure design)

The Roles of SA & SD have been shown schematically.

→ During structured Analysis, the 'SRS' document is transformed into a "data flow diagram (DFD) model.

→ During 'SD', the 'DFD' model is transformed into a structure chart.



→ It is imp" to understand that the purpose of structured Analysis is to capture the detailed Structure of the system as perceived by the user, whereas the purpose of (SD') is to define the structure of the solution that is suitable for implementation in some programming language.

Structured Analysis :-

A 'DFD' is a hierarchical graphical model of a system that shows the diff" processing activities.

Principles :-

→ Top-down decomposition approach

→ Application of divide & conquer principle. Through this each high-level function is independently decomposed into detailed functions.

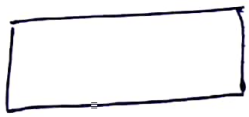
→ Graphical representation of the Analysis results using data flow diagrams (DFDs)

Data flow Diagrams (DFD's)

The DFD (also known as the bubble chart) is a simple graphical formalism that can be used to represent a system in terms of the input data to the system various processing carried out on those data, & the output data generated by the system.

A DFD model represents the the sub-functions performed by the functions using a hierarchy of diagrams. A DFD model uses a very limited no. of primitive symbols.

primitive symbols used for constructing DFD's:

1.  → External entity
2.  → process
3. → Dataflow
4.  output
5. ═══ (Data store)

Function Symbol (○) :- A function is represented using a circle.

External Entity (▭) :-

An External Entity such as a librarian, a library member.

Data flow (→) :-

A directed arc (arrow) is used as a data flow symbol. a data flow symbol represents the data flow occuring b/w 2 processes.

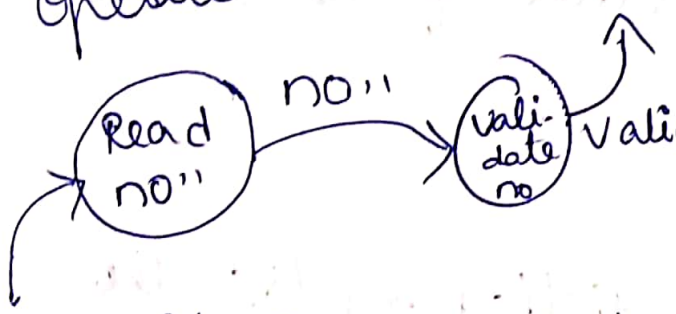
Data store (═) : it is represented using 2 parallel lines. it represents a logical file. (i.e A data store symbol can represent either a data structure (or) physical file on disk.

Output (▭) : used to when hard copy is produced.

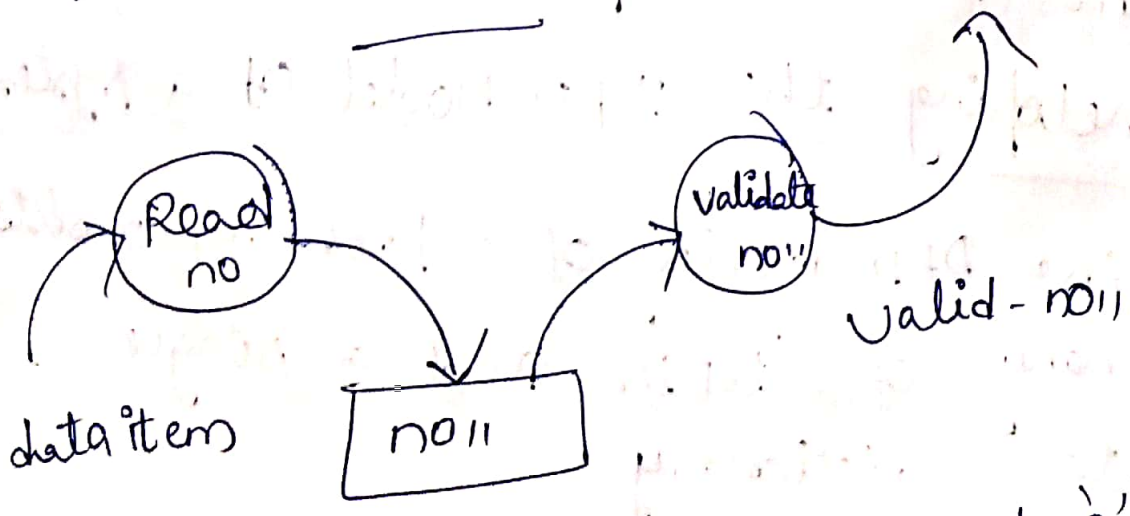
Important concepts associated with constructing DFD. models :-

Synchronous & Asynchronous operations

If 2 bubbles are directly connected by a data flow arrow, then they are synchronous. This means that they operate at the same speed.



a) Synchronous operation of 2 bubbles.



b) Asynchronous operation of 2 bubbles.

→ Here the "validate-no" bubble can start processing only after the 'Read-no' bubble has supplied data to it. and the "read-no" bubble has to wait until the "validate-no" bubble has consumed its data.

However, if 2 bubbles are connected through a data store, then the speed of operation of the bubbles are independent.

Data Dictionary:-

A data dictionary lists the purpose of all data items and the definition of all composite data items in terms of their component data items.

Developing the DFD Model of a system

The DFD model of a problem consists of many of 'DFD's and a single data dictionary.

Content Diagram:

The content diagram is the most abstract (highest level) data flow representation of a system.

The content diagram establishes the content in which the system operates. (i.e who are the users, what data do they input to the system, and what data they received by the system.

Decomposition:

Each bubble in the DFD represents a function performed by the system. The bubbles are decomposed into sub-functions at the successive levels of the DFD model.

Numbering of bubbles:-

it is necessary to no. of diff. bubbles occuring in the DFD. these nos. help in uniquely identifying any bubble in the DFD from its bubble no.

The bubble at the content level is usually assigned the no., 0 to indicate that it is the 0 level DFD. bubbles at level 1 are no.1, 0.1, 0.2, 0.3 etc.,

Balancing DFD's

The data that flow into (or) out of a bubble must match the data flow at the next level of DFD. this is known as balancing a 'DFD'.

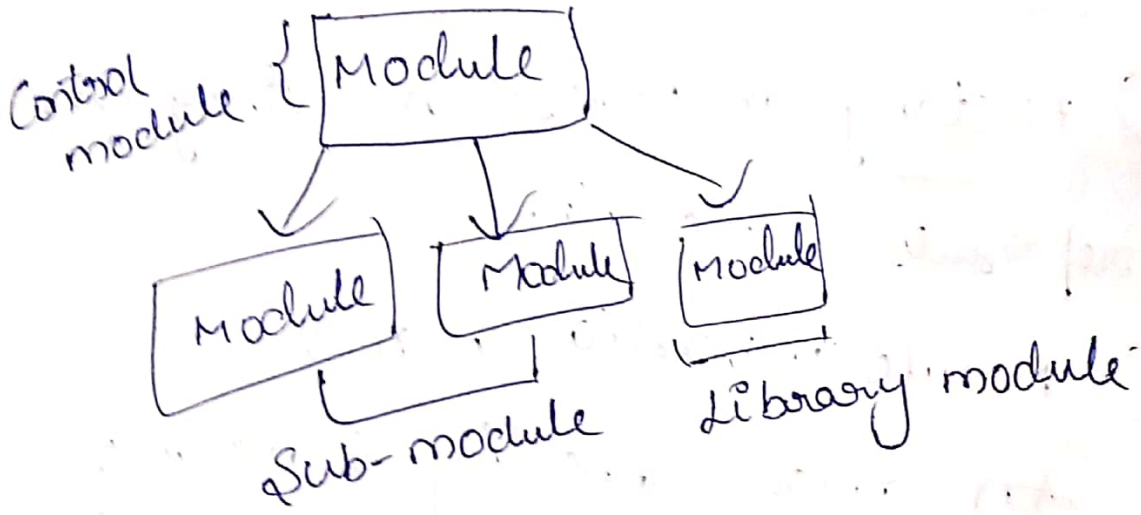
Structure Charts:

Structure chart is a chart derived from 'Data flow diagram'. it represents the system in more detail than DFD. it breaks down the Entire System into lowest functional modules.

→ Structure chart Represents hierarchical structure of modules. At each layer a specific task is performed.

→ Here are the symbols used in construction of structure chart

→ Module = It Represents process.
A control module branches to more than one sub-module. Library modules are re-usable & invokable from.



Structured design:

The aim of structured design is to transform the results of the structured Analysis into a "structure Chart". A structure chart represents the S/W Architecture. The various Modules making up the system, the module dependency & the parameters that are passed among the diff. modules. The structure chart representation can be easily implemented using some programming languages.

Basic Building blocks using which Structure charts are designed as follow:

(1) Rectangular Boxes: A rectangular box represents a module.

(2) Module invocation Arrows:
An arrow connecting 2 modules implies that during program execution control is passed from one module to another module.

(3) Data flow arrows: These are small arrows appearing alongside the module invocation arrows.

(4) Library Module: It is usually represented by a rectangle with double edges.

(5) Selection: The diamond symbol represents the fact that one module of several modules connected with the diamond symbol.

(6) Repetition :- A loop around the control flow arrows denotes that the respective modules are invoked repeatedly.

Flow Chart Vs Structure Chart :-

→ A flow chart representation of a program. it is a technique to represent the flow of control in a program.

→ Structure Chart in 3 principle ways

(i) it is usually difficult to identify the diff. modules of a program from its flow Chart representation.

-) Data interchange among diff. modules is not represented in a flow chart.

→ Sequential ordering of tasks that is inherent to a flow chart is suppressed in a Structure Chart.

Transformation of a DFD model into Structure Chart:-

Systematic techniques are available to transform the DFD representation of a problem into a module structure represented by as a Structure chart".

SD provides 2 strategies:-

(i) Transform Analysis

(ii) Transaction Analysis

(i) Transform Analysis: It identifies the primary functional components / modules & the input & o/p data for these components It divide the DFD into 3 types

→ Input → output.

→ processing

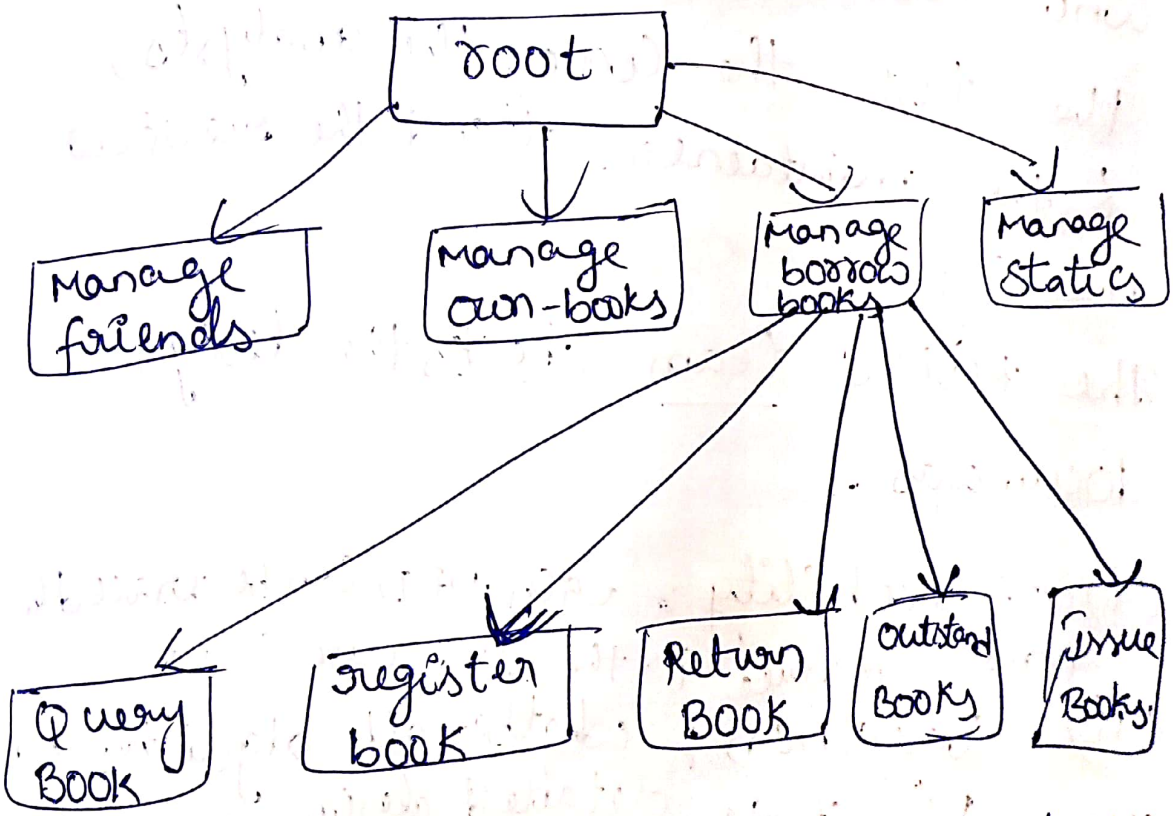
(ii) Transaction Analysis: it is an alternative to transform analysis and is useful while designing transaction process programs.

A transaction allows the users to perform some specific type of work by using the s/w.

Ex:-

Issue Book, Return Book, Query Book Etc.. are transactions.

Ex: Draw the Structure Chart for the personal library s/w.



Detailed Design:- During detailed design the "Pseudo Code description of the processing & the diff" "DS" are designed for the diff" modules of the Structure Chart.

Design Review:-

After a design is complete, the design is required to be reviewed. The review team usually consists of members with design, implementation, testing & maintenance. who may/may not be the members of the development team. normally, members of the team who could code the design, and the test the code, the analysts, & the maintainers attend the review meeting.

The Review team checks the design documents :-

(i) Traceability :- Each DFD can be traced to some module in the structure chart.

(ii) Correctness :- whether all algorithms and DS of the detailed design are correct

(iii) Maintainability :- design can be easily maintained in future.

(iv) Implementation :- whether the design can be easily & efficiently be implemented

Types of DFD:

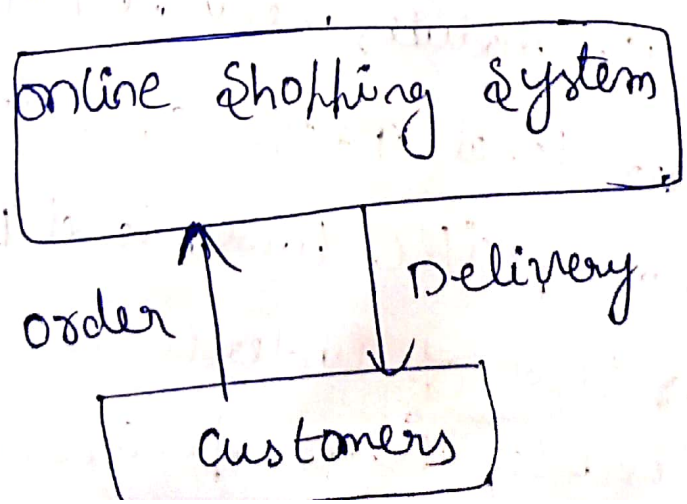
(i) Logical DFD: This type of DFD concentrates on the system process & flow of data in the system.

Ex: In a Banking S/W system how data is moved b/w diff" Entities.

(ii) Physical DFD:- This type of DFD shows how the data flow is actually implemented in the system.

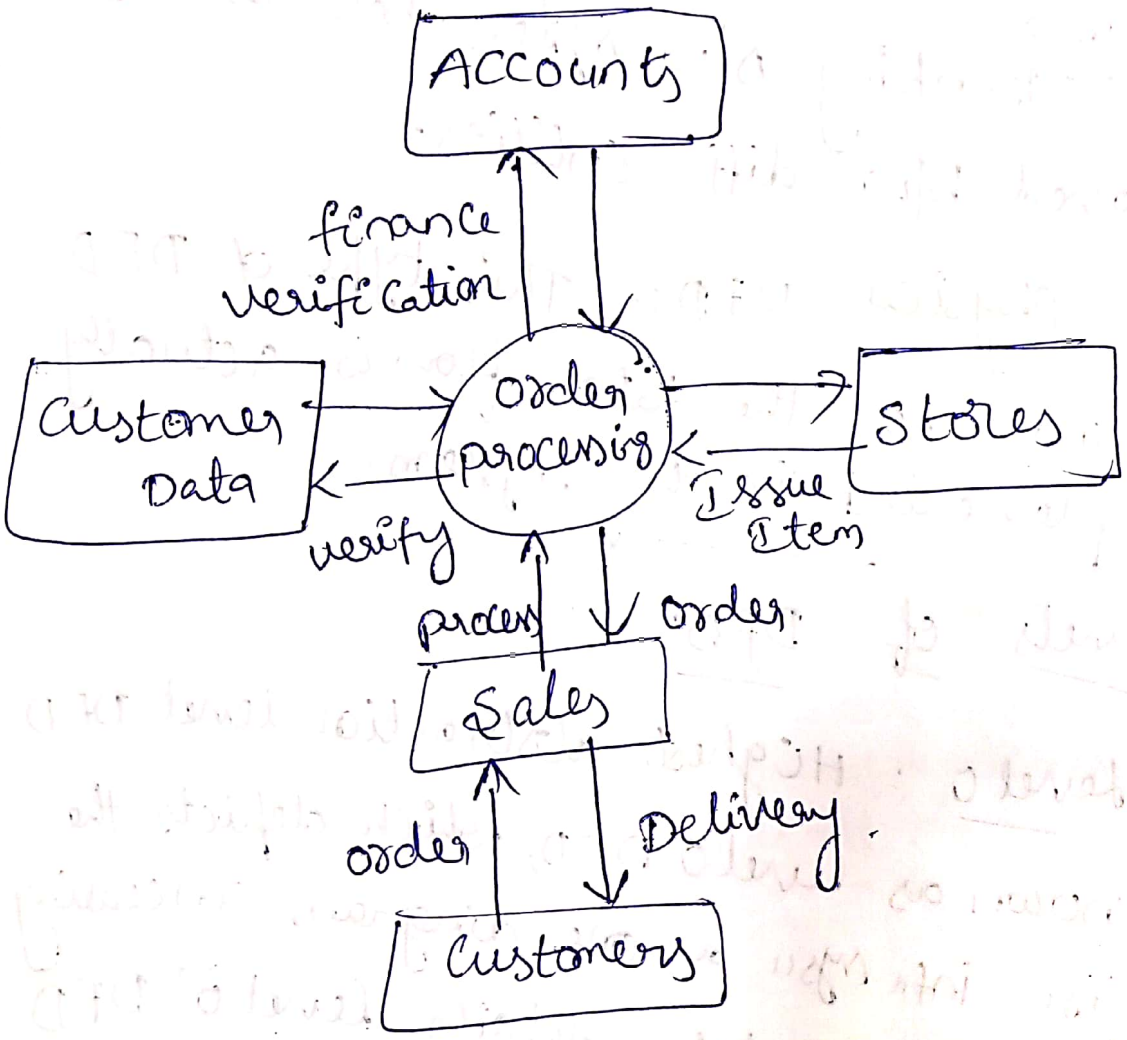
Levels of DFD:

① Level 0 : Highest abstraction level DFD is known as level 0 DFD, which depicts the entire inf" sys" as one diagram concealing all the underlying details. Level 0' DFD are also known as Context level DFD's.



Level 1 :-

The level 0 DFD is broken down into more specific, this depicts basic modul in the system & flow of data among various modules.



Level 2 :- In this DFD shows how data flow inside the modules mentioned in level1 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.

Structured English

other forms of methods, which use graphs.

Hence, Analysts & designers of the S/W come up with tools such as Structured English. It is nothing but the description of what is required to code & how to code it. Structured English helps the programmer to write error-free code.

Pseudo-code :- It is written close to programming language. Pseudo-code contains more programming details than structure English. It provides a method to perform the task. If a computer is executing the code.

Decision Tables :- It represents conditions and the respective actions to be taken to address them.

Object - Oriented design:

(1) **Object**: All Entities involved in the solution design is known as object.

Ex: person, banks, company.

(2) **class**:- A class is a generalized description of an object. An object is an instance of a class. A class defines all attributes, functions & methods.

(3) **Message**:- objects communicate by message passing.

(4) **Abstraction**: Complexity is handled using Abstraction.

(5) **Encapsulation**:- Info. hiding.

(6) **Inheritance**:- OOD allows similar classes to stack up in a hierarchical manner where the lower.

(7) **Polymorphism**:- OOD languages provide a mechanism where methods performing similar tasks.

UML :- (Unified Modeling language)

* visualizing
* constructing
* specifying
* Documenting.

Application Domains of UML :-

(i) Enterprises
(ii) Bank & financial services
(iii) Telecommunication
iv) Transportation & Defense

Conceptual Model of UML :-

(i) Building blocks of UML
(ii) Rules
(iii) Common mechanisms.

Building Blocks of UML :- These are the fundamental Elements in UML.

(i) Things
(ii) Relationship
(iii) Diagrams.

Things :- A diagram can be viewed as a graph containing vertices & Edges. Vertices are replaced by things. Edges are replaced by relationship.

(i) Structural Things

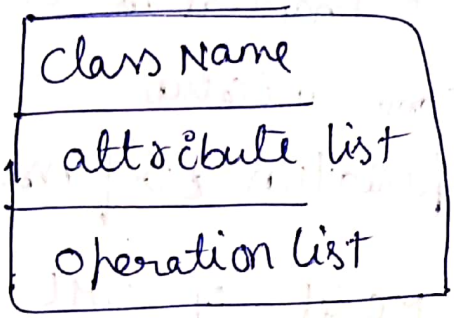
ii) Behavioral "

iii) Grouping "

iv) Annotational "

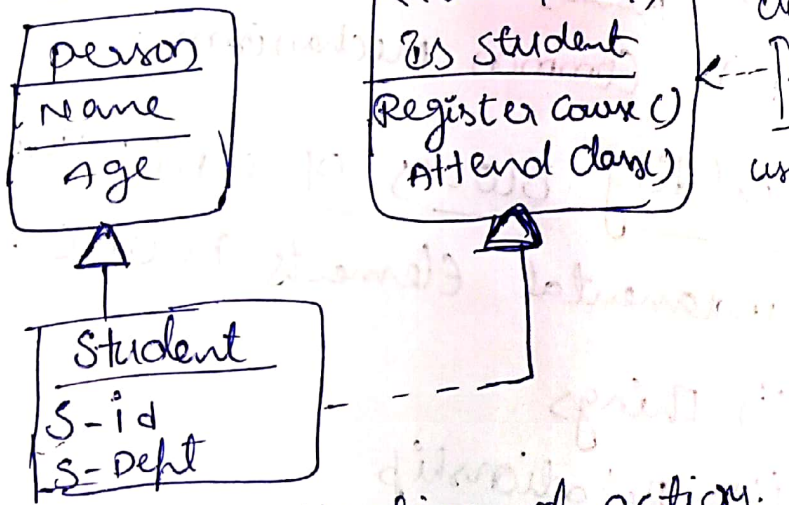
(i) Structural Things: (7)

→ Class:

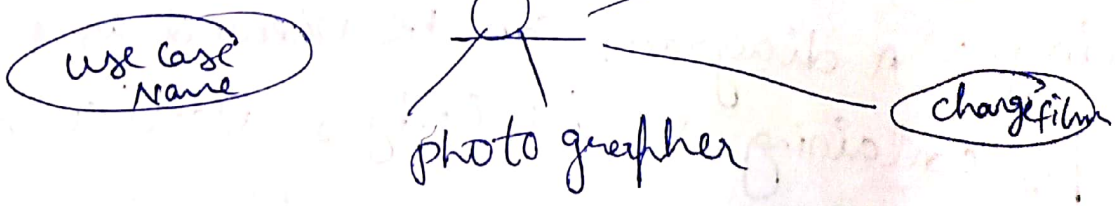


Interface:- collection of operation signatures represented by a circle.

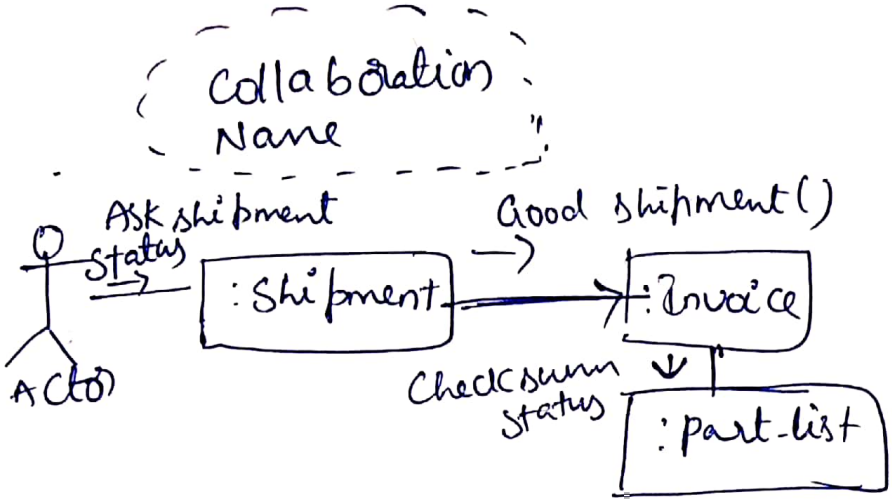
Interface Name



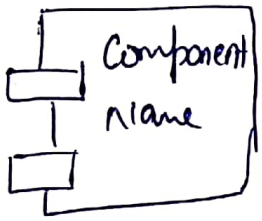
use case:- It is a collection of actions.



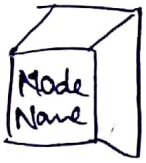
Collaboration: It is a collection of interacting



Component: A component is a physical & replaceable part of a system. It is represent by "tabbed Rectangle"



Node :- A Node is a physical element that exist at run time.



Active class:- A class whose object can initiate its own flow of control and works in parallel with other objects.