Unit 5

Software Design

Unit Outcomes (UOs) :

After completion of this unit, students will be able to :

5.1 Prepare software design

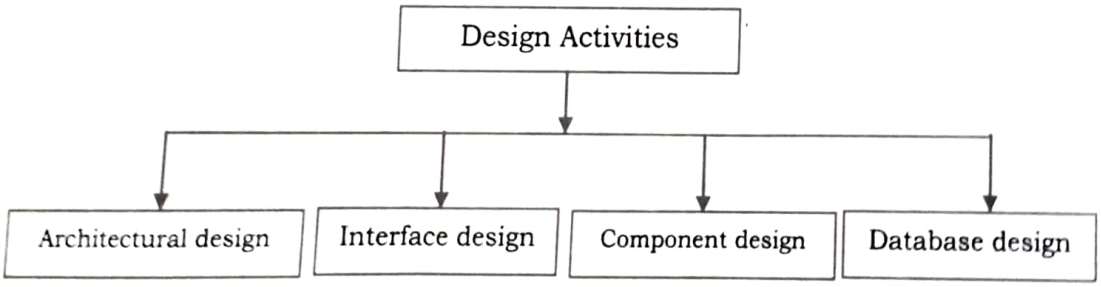
5.2 Design proper user interface for the software

5.1.1 Software design process

- The design process is a sequence of steps to describe all aspects of the software.
- Software design process specifies 'how' aspect of the system (means how the system will be implemented and how it will work).
- The purpose of the software design process is → to plan a solution of problems specified in SRS.
- It includes: user interface design, input output design, data design, process and program design and technical specification etc.
- It transforming the customer requirements (described in SRS) into appropriate form that is suitable to implement using any of programming languages.
- Output of Software design process is →**design documents**.
- Two main aspects of software design process are :
  - ✓ *Design activities*
  - ✓ *Design methodologies*

❖ Design activities

- Design activities are depending on the type of the software being developed.
- The classification of design activities are given below :



Design Activities

1. Architectural design

- Where you can identify the overall structure of the system, sub-systems, modules and their relationships.

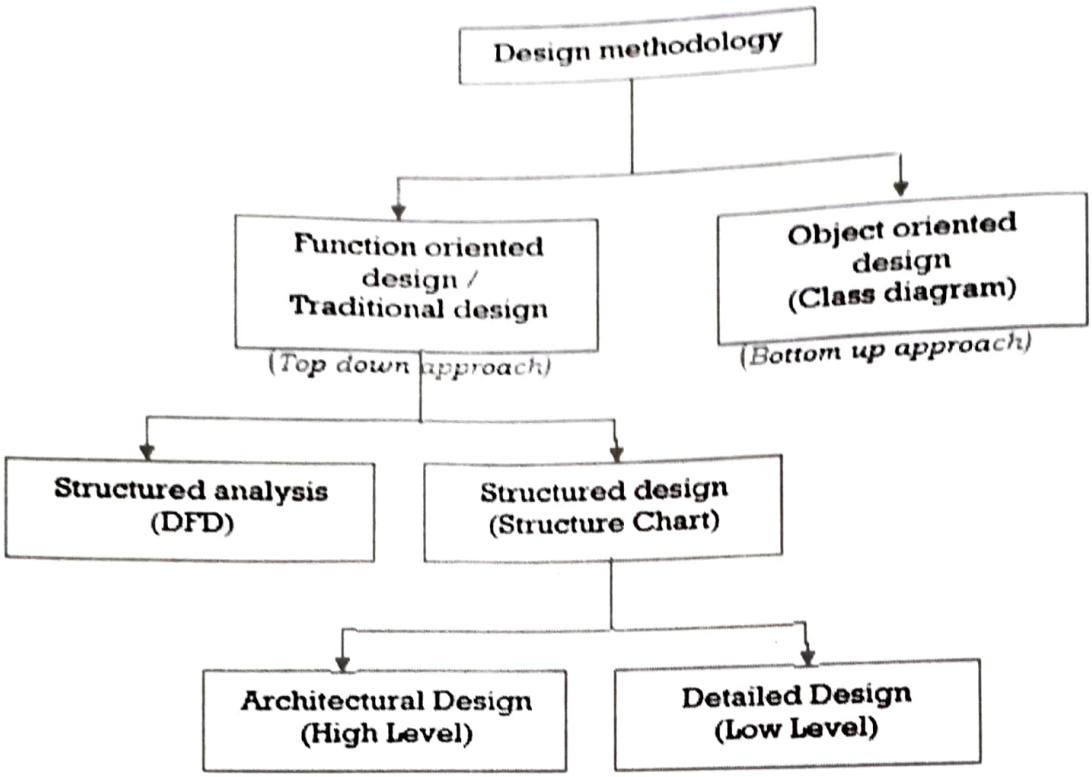
- It defines the framework of the computer based system.
- It can be derived from DFD (data flow diagram).

2. **Interface design**
   - Where you can define the interface between system components.
   - It describes how system communicates with itself and with the user also.
   - It can be derived from DFD and State transition diagram.
3. **Component design**
   - That defines each system component and show how they operate.
   - It can be derived from State transition diagram.
4. **Database design**
   - Where you can define the system data structure and show how they are represented in a database.
   - Existing database can be reused or a new database to be created.
   - The data objects and their relationships are defined in ERD (entity relationship diagram) and detailed content described in data dictionary (DD).
   - It can be derived from ERD and DD.

Design methodologies

- Design methodologies are followed in software development from beginning up to the completion of the product.
- Design methodologies use to provide guidelines for the design activity.
- The nature of the design methodologies are dependent on the following factors:
  - The software development environment.
  - The type of the system being developed.
  - User requirements.
  - Qualification and training of the development team.
  - Available software and hardware resources.
- There are large numbers of software design methodologies. Different methodologies are used to solve different type of problems.

Classification of design methodologies is shown in the figure.



Design methodologies

- There are fundamentally two different approaches:

1. Function oriented design
2. Object oriented design

1. Function oriented design

- In which the set of functions are described.
- Starting at this high-level view of the system, each function is successively refined into more detailed functions or sub functions *(top down approach)*.
- Each of these sub-functions may be split into more detailed sub functions and so on.
- Function oriented design further classified into →Structure analysis and Structure design.

➥ **Structure analysis**

- It is used to transform a textual description into graphical form.
- It examines the detail structure of the system.
- It identifies the processes and data flow among these processes.
- In structure analysis – functional requirements specified in SRS are decomposed and analysis of data flow is represented here.
- Structure analysis is graphically represented using data flow diagram (DFD).

➥ **Structure design**

- During structured design, the results of structured analysis are transformed into the software design.

- All the functions identified during analysis are mapped to module structure and that is useful for implementation.
- The aim of the structure design is → to transform the result of the structure analysis (DFD) into a structure chart. So structure analysis is graphically represented using structure chart.
- Two main activities are performed during structure analysis:

i. ***Architecturaldesign (High-level design)*** – decomposing the system into modules and build relationship among them.

ii. ***Detailed design (Low-level design)*** – individual modules are design with data structure and algorithms.

- Problem with the structure design is that → if a change is made in one part of the program will require search through of entire program.

2. **Object oriented design**

- In this design the system is viewed as a collection of objects (entities).
- Objects available in the systems are identified and their relationships are built during this approach. And the whole system is built (bottom up approach).
- Each object has its own internal data and similar objects constitute a class. So each object is a member of a class.
- Class diagram is used to represent the object oriented design. UML modeling and use case are also used in object oriented design.
- Data in the system is not centralized and shared but is distributed among the objects in the system.

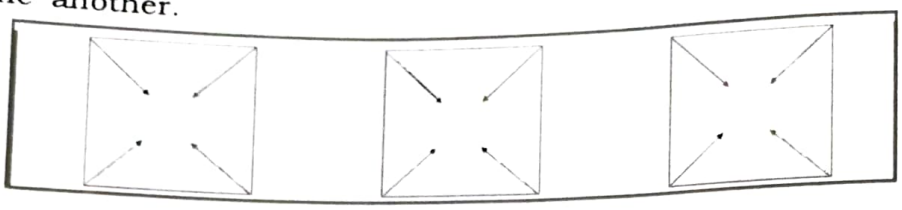
5.1.2 Cohesion and Coupling

- Modularity is clearly a desirable property of any software development.
- In software development, modularity is → decomposition of a program into smaller programs (or modules).
- A system is considered *modular* if it consists of multiple modules so that each module can be implemented separately and debugged separately.
- Modular system provides advantages like:
  - Easy to understand the system.
  - System maintenance is easy.
  - Provide reusability.
- Modularity is successful because developers use prewritten code, which saves resources. Overall, modularity provides greater software development manageability.
- Cohesion and coupling are two modularization criteria that are often used together.
- Most researchers and developers are agreed that for good software design neat decomposition is highly needed, and the primary characteristic of neat decomposition is***'high cohesion and low coupling'.***

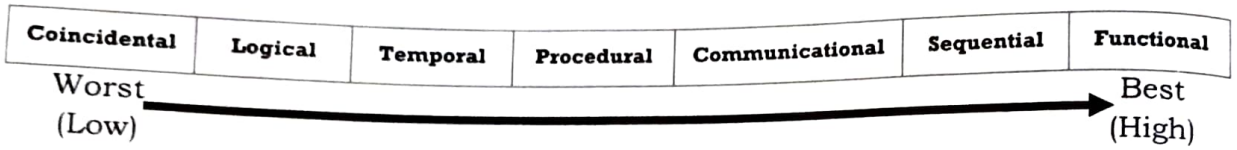
Cohesion :

- Cohesion is → a measure of functional strength of a module.
- Cohesion keeps the internal modules together, and represents the functional strength.

- Cohesion of a module represents how tightly bound the internal elements of a module are to one another.



➥ **Classification of cohesion**

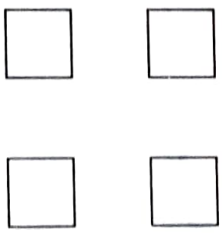


- **Coincidentalcohesion**
  - It is the lowest cohesion. Coincidental cohesion occurs when there are no meaningful relationships between the elements.
  - A module is said to have coincidental cohesion, if it performs a set of tasks that relate to each other very loosely.
  - It is also called random or unplanned cohesion.
- **Logicalcohesion**
  - A module is said to be logically cohesive if there is some logical relationships between the elements of module, and the elements perform functions that fall into same logical class.
  - For example: the tasks of error handling, input and output of data.
- **Temporal cohesion**
  - Temporal cohesion is same as logical cohesion except that the elements are also related in time and they are executed together.
  - A module is in temporal cohesion when a module contains functions that must be executed in the same time span.
  - Example: modules that perform activities like initialization, cleanup, and start-up, shut down are usually having temporal cohesion.
- **Proceduralcohesion**
  - A module has procedural cohesion when it contains elements that belong to common procedural unit.
  - A module is said to have procedural cohesion, if the set of the modules are all part of a procedure (algorithm) in which certain sequence of steps are carried out to achieve an objective.
  - Example: the algorithm for decoding a message
- **Communicationalcohesion**
  - A module is said to have communicational cohesion, if all functions of the module refer to or update the same data structure, for example the set of functions defined on an array or a stack.
  - These modules may perform more than one function together.

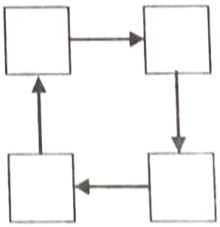
- **Sequentialcohesion**
  - When the output of one element in a module forms the input to another, we get sequential cohesion.
  - Sequential cohesion does not provide any guideline how to combine these elements into modules.
  - For example, in a TPS (transaction processing system), the get-input, validate-input, sort-input functions are grouped into one module.
- **Functionalcohesion**
  - Functional cohesion is the strongest cohesion.
  - In it, all the elements of the module are related to perform a single task.
  - All elements are achieving a single goal of a module.
  - Functions like: compute square root and sort the array are examples of these modules.

Coupling

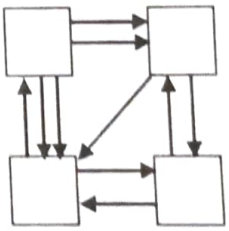
- Coupling between two modules is →a measure of the degree of interdependence or interaction between these two modules.
- Coupling refers to the number of connections between 'calling' and a 'called' module. There must be at least one connection between them.
- It refers to the strengths of relationship between modules in a system. It indicates how closely two modules interact and how they are interdependent.
- As modules become more interdependent, the coupling increases. And loose coupling minimize interdependency that is better for any system development.
- If two modules interchange large amount of data, then they are highly interdependent or we can say they are highly coupled.
- High coupling between modules make the system difficult to understand and increase the development efforts. So low (OR loose) coupling is the best.



No coupling



Loose coupling



High coupling

Classification of coupling

Five different types of coupling can occur between two modules.

Data	Stamp	Control	Common-	Content

Best (Low) → Worst (High)

Data coupling

- Two modules are data coupled, if they communicate using an elementary data item that is passed as a parameter between these two.
- For example → an int, a char, a float etc.
- It is lowest coupling and best for the software development.

Stamp coupling

- Two modules are stamp coupled, if they communicate using a composite data item such as a record in PASCAL or a structure in C.

Control coupling

- Control coupling exists between two modules, if data from one module is used to direct the order of instructions execution in another module.
- An example of control coupling is a flag set in one module and tested in another module

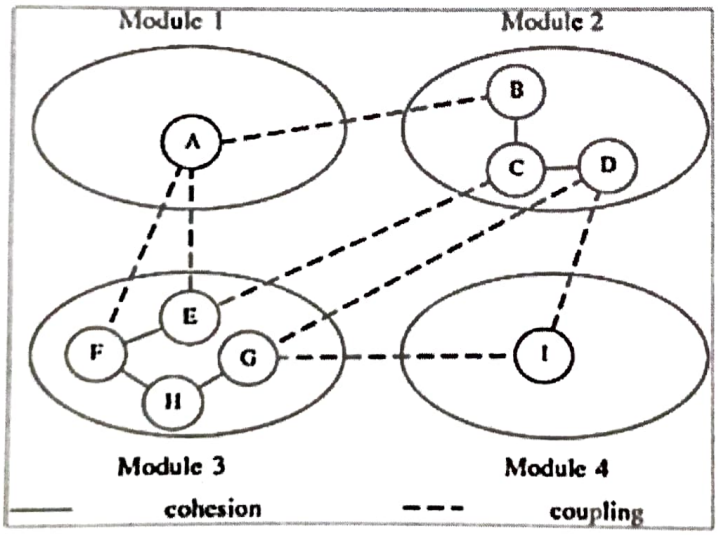
Common coupling

- Two modules are common coupled, if they share data through some global data items. It means two or more modules are communicating using common data.

Content coupling

- It is the highest coupling and creates more problems in software development.
- Content coupling exists between two modules, if they share code, e.g. a branch from one module into another module.
- It is also known as 'pathological coupling'.

Cohesion and Coupling can be shown using following figure:



Functional independence

- A module having high cohesion and low coupling is said to be functionally independent of other modules.
- So, that a cohesive module performs a single task or function.
- A functionally independent module has minimal interaction with other modules.

➢ ***Intra dependency (Cohesion) between modules should be high and inter dependency (Coupling) should be low.***

For good software design neat decomposition is highly needed, and the primary characteristic of neat decomposition is '***high cohesion and low coupling***'.

Need of functional independence

- Functional independence is a good key to any software design process due to following reasons:

1. Error isolation

- It reduces error propagation.
- The reason behind this is → if a module is functionally independent, its degree of interaction with the other modules is less.
- So the error of one module can't affect other module.

2. Scope of reuse

- Reuse of a module becomes possible. Because each module does some well-defined and precise function, and the interaction of the module with the other modules is simple and minimal.
- Therefore, a cohesive module can be easily taken out and reused in a different program.

3. Understandability

- Complexity of the design is reduced, because different modules can be understood in isolation as modules are more or less independent of each other.

Difference between Cohesion and Coupling

Cohesion	Coupling
→ Cohesion is the indication of the relationship within module.	→ Coupling is the indication of the relationships between modules.
→ Cohesion shows the module's relative functional strength.	→ Coupling shows the relative interdependence among the modules.
→ Cohesion is a degree (quality) to which a component / module focuses on the single thing.	→ Coupling is a degree to which a component / module is connected to the other modules.
→ While designing you should go for high cohesion.i.e. a cohesive component/ module focus on a single task with little interaction with other modules of the system.	→ While designing you should go for low coupling i.e. dependency between modules should be less.
→ Cohesion is the kind of natural extension of data hiding for example, class having all members visible with a package having default visibility.	→ Making private fields, private methods and non-public classes provides loose coupling.
→ Cohesion is Intra Module Concept.	→ Coupling is Inter -Module Concept

5.1.3 Approach of software design

In the approach of software design, the system is divided in to sub-systems and each sub-system or component is then treated as a system and decomposed further.

❖ **Function oriented design**

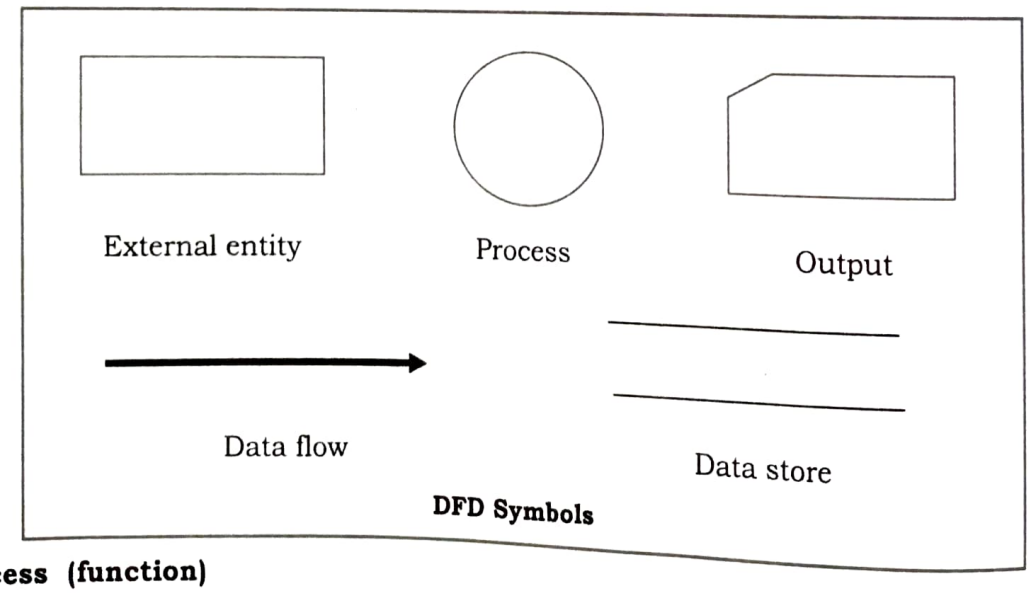
- In it, system is describes as a set of functions.
- Starting at this high-level view of the system, each function is successively refined into more detailed functions or sub functions *(top down approach)*.
- Each of these sub-functions may be split into more detailed sub functions and so on.
- Data in the system is centralized and shared among different functions.

❖ **Data flow diagrams**

- DFD (Data Flow Diagram) is also known as bubble chart or data flow graph.
- DFDs are very useful in understanding the system and can be effectively used during analysis. It shows flow of data through a system visually.
- The DFD is a hierarchical graphical model of a system that shows the different processing activities or functions that the system performs and the data interchange among these functions.
- It views a system as a function that transforms the inputs into desired outputs.
- Each function is considered as a process that consumes some input data and produces some output data.
- The system is represented in terms of the input data to the system, various processing carried out on these data, and the output data generated by the system.
- Functional model can be represented using DFD.

➥ **Primitive symbols used in construction of DFD model**

- The DFD model uses a very limited number of primitive symbols.



DFD Symbols

1. **Process (function)**
   - Process or function is represented by circle or bubble.
   - Circles are annotated with names of the corresponding functions.
   - A process shows the part of the system that transforms inputs into outputs.

- The process is named using a single word that describes*what* the system does functionally. Generally process is named using '***verb***'.

2. **External entity**
   - Entity is represented by a rectangle. Entities are external to the system which interacts by inputting data to the system or by consuming data produced by the system.
   - It can also define source (originator) or destination (terminator) of the system.
3. **Data flow**
   - Data flow is represented by an arc or by an arrow.
   - It used to describe the movement of the data.
   - It represents the data flow occurring between two processes, or between an external entity and a process. It passes data from one part of the system to another part.
   - Data flow arrows usually annotated with the corresponding data names. Generally data flow named using '***noun***'
4. **Data store**
   - Data store is represented by two parallel lines.
   - It is generally a logical file or database.
   - It can be either a data structure or a physical file on the disk.
5. **Output**
   - Output is used when a hardcopy is produced.
   - It is graphically represented by a rectangle cut either a side. (in figure 2.3)

➥ **Developing DFD model of the system**

- DFD graphically shows the transformation of the data input to the system to the final result through a hierarchy of levels.
- DFD starts with the most abstract level of the system (lowest level) and at each higher level, more details are introduced.
- To develop higher level DFDs, processes are decomposed into their sub functions.
- The abstract representation of the problem is also called *context diagram*.

➢ **Context diagram (Level 0 DFD)**

- The context diagram is top level diagram; it is the most abstract data flow representation of a system.
- It is an overall, simplified view of target system.
- It only contains one process node that generalizes the function of entire system with external entities. (It represents the entire system as a single bubble.)
- Data input and output are represented using incoming and outgoing arrows. These arrows should be annotated with the corresponding data items (usually a noun).

➢ **Level 1 diagram**

- To develop the level 1 DFD, we have to examine the high-level functional requirements
- It is recommended that 3 to 7 functional requirements can be directly represented as bubbles in 1st level.

➤ **Further Decomposition (Level 2 and above)**

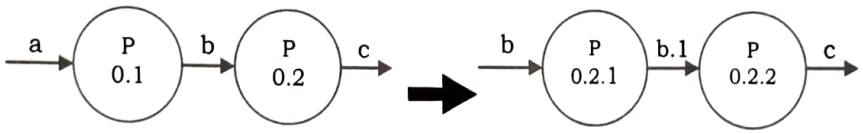
- The bubbles are decomposed into sub-functions at the successive levels of the DFD.
- Decomposition of a bubble is also known as *factoring or exploding* a bubble.
- Each bubble at any level of DFD is usually decomposed between 3 to 7 bubbles in its higher level.
- Successive levels give more detailed description about the system.
- It's not a rule that particular number of levels are needed for the system, but decomposition of a bubble should be carried on until a level is reached at which the function of the bubble can be described using a simple algorithm.

➥ **Numbering the bubbles**

- It is necessary to number the different bubbles occurring in the DFD.
- These numbers help in uniquely identifying any bubble in the DFD by its bubble number.
- The bubble at the context level is usually assigned the number 0 to indicate that it is the 0 level DFD.
- Bubbles at level 1 are numbered, 0.1, 0.2, 0.3, etc.

➥ **Balancing DFD**

- 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.



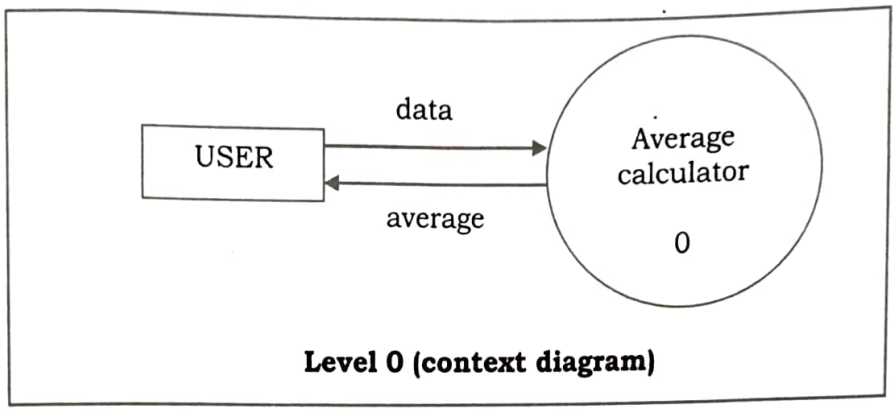
Data flow in level - 1 **Data flow in level - 2**

➥ **Some care should be taken while constructing DFDs (Guidelines when drawing DFDs)**

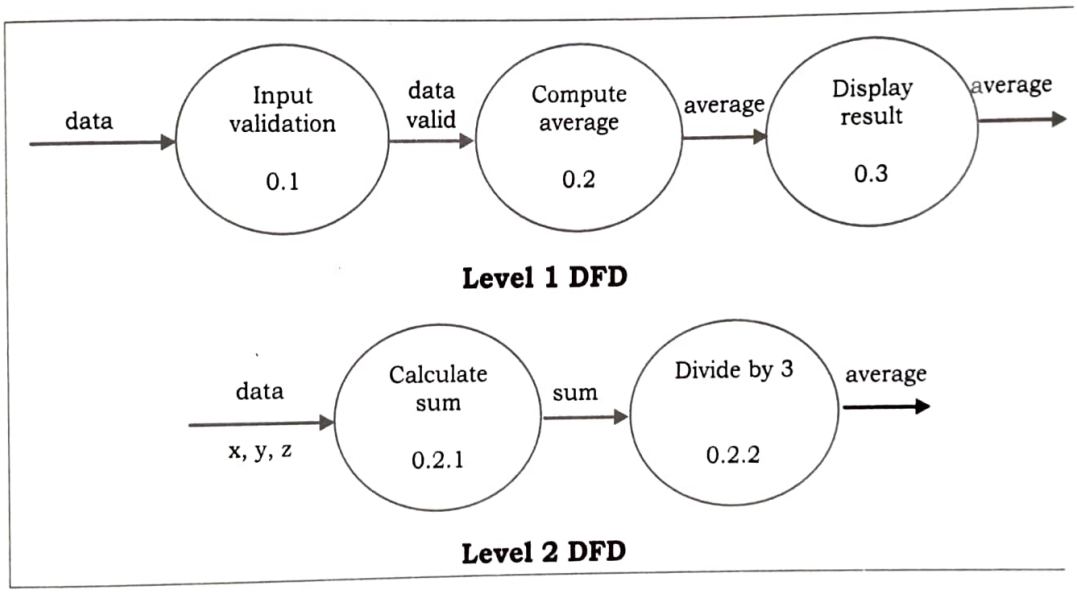
- A process must have at least one input and one output data flow.
- No control information (IF-THEN-ELSE) should be provided in DFD.
- A data store must always be connected with a process. A data store cannot be connected to another data store or to an external entity.
- Data flows must be named.
- Data flows from entities must flow into processes, and data flows to entities must come from processes.
- There should not be detailed description of process in context diagram.
- Name of the data flow should be noun and name of process should be verb.
- Each low level DFD must be balanced to its higher level DFD (input and output of the process must be matched in next level).
- Data that travel together should be one data flow.
- No need to draw more than one bubble in context diagram.

- Generally all external entities interacting with the system should be represented only in the context diagram.
- Be careful with number of bubbles in particular level DFD, as too less or too many bubbles in DFD are oversight.
- All the functionalities of the system specified in SRS must be captured by the DFD model.

Simple DFD example of average calculator of three numbers.



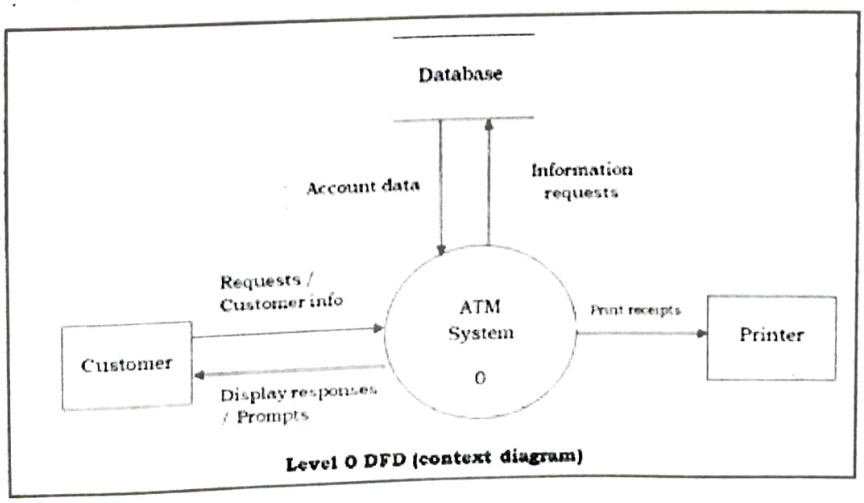
Level 0 (context diagram)



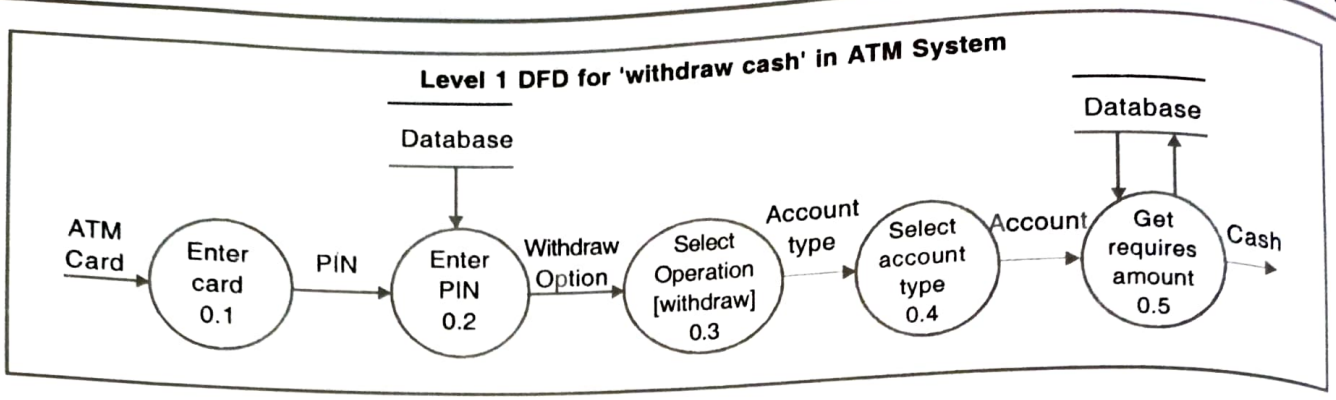
Level 1 DFD

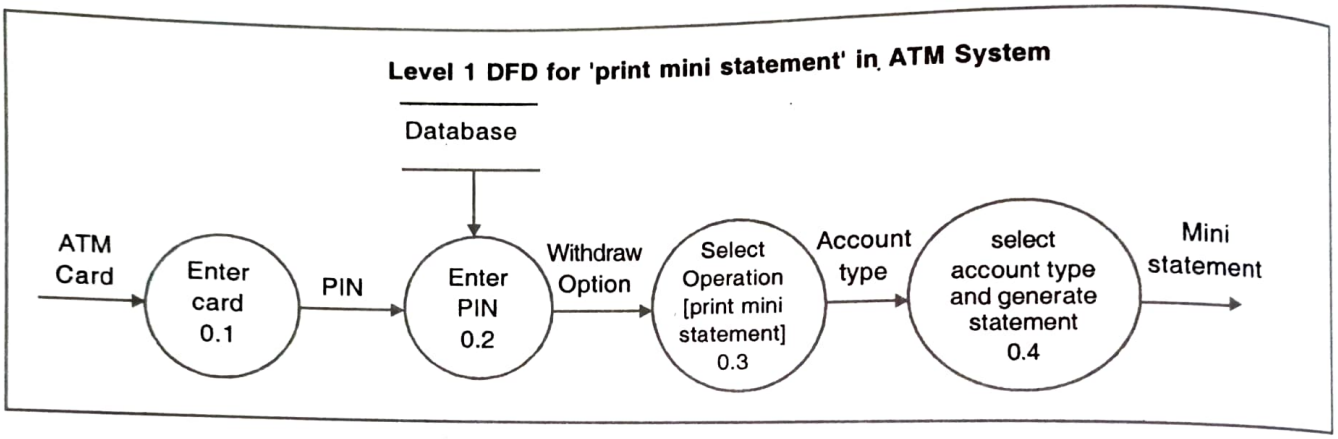
Level 2 DFD

Another example of ATM system



Level 0 DFD (context diagram)





Note: *Level1 DFD can be drawn for each high level functional requirement shown in the SRS.*

Advantages of DFD model

- DFD is a simple graphical technique which is very simple to understand and easy to use.
- It can be used as a part of system documentation.
- DFD can provide detailed description of the system components.
- It provides clear understanding to the developers about the system boundaries and analysis of the system.
- It explains the logic behind the data flow within system.
- It provides structure analysis of the system.
- Symbols used in DFD model are very less.
- It does not provide any time dependent behaviour like we cannot consider at which time we have to do particular process.

Disadvantages of DFD model

- Control information is not defined by a DFD.
- DFD does not provide any specific guidance as how exactly decompose a given function into its sub functions; we have to use subjective judgment to carry out decomposition.
- Sometimes it puts programmers in little confusing state.
- Different DFD models have different symbols, e.g. in Gane and Sarson notations → process is represented as rectangle while in Demarco and Yordan notations → it is ellipse, so making confusion at the time of referring. (*in some of the notations you can see*

the process symbol is rectangle while in some other notations, process symbol is ellipse and that is confusing).

- Physical considerations are left out in DFD.

Data dictionary

- Data Dictionary is the major component in the structured analysis model of the system.
- DD lists all the data items appearing in DFD.
- A data dictionary is a file or a set of files that contains a database's metadata.

> ***Note :*** Metadata is data about data. In other term → it is set of data used to describe the properties and characteristics of some other data.)

- It is storing all up-to-date information about the objects like tables, columns, index, constraints, functions etc.

The need of data dictionary

- It describes the meanings and purposes of data elements within the context of a project.
- To provide detailed information about the contents of a dataset or database.
- It also provides metadata about data elements.
- Helps to design use case and designing the software.
- It allows others to understand your data.

The data dictionary contains the following information:

- Names of all the tables and schemas present in the database.
- Description of the database users.
- Capture the relationships between data items.
- Contains aliases of the data items.
- Range of all the possible values to be inserted.

Types of data dictionary

Mainly two types of data dictionary are there:

(i) Active data dictionary

- If data dictionary is managed automatically by database management software, it is called active data dictionary.
- It is also called integrated data dictionary.
- It is always consistent with the current updates with the database system.

(ii) Passive data dictionary

- If data dictionary is managed by the users of the system, it is called passive data dictionary.
- It is also called non-integrated data dictionary.
- It is not consistent with the current update of the database, manual changes require to update.

Advantages of data dictionary

- It gives the well-structured and clear information about the system database.

- It is very helpful for the administrator or system analyst to understand the system database.
- It helps in improving the communication between the user and the system analyst.
- It helps in designing test cases and designing the software.
- The data dictionary is also important to find the specific data item object from the list.

➥ **Disadvantages of data dictionary**

- High cost to build up.
- Hard to understand for non-technical users.
- It does not provide functional details.
- Creating a new data dictionary is complex task.

A simple example of data dictionary outlining a person table in database.

Data item (Attribute name)	Data type	Data format	Size	Constraint	Description	Example
First_Name	String	--	25	Not null	First name of the user	AARUSH
Last_Name	String	--	25	--	Last name (surname) of the user	PARMAR
Bdate	Date	dd-mm-yyyy	08	--	Birth date	15-01-2015
Roll_no	Number	--	10	Primary key	Roll number of the user	1234567890
City	String	--	15	--	City of the user	surendranagar

❖ Object oriented design

- In this design the system is viewed as a collection of objects (entities).
- Objects available in the systems are identified and their relationships are built during this approach. And the whole system is built (bottom up approach).
- Each object has its own internal data and similar objects constitute a class. So each object is a member of a class.
- Class diagram is used to represent the object oriented design. UML modeling and use case are also used in object oriented design.
- Data in the system is not centralized and shared but is distributed among the objects in the system.
- Three main concepts: *Encapsulation, Inheritance and Polymorphism* greatly enhance the ability of developers' to more easily manage the software product.
- *Encapsulation*combining data and functions into a single entity, *inheritance* provide reusability and by *polymorphism* same context can be used for different purposes.

➥ **Advantages of object oriented design**

- Reduce maintenance.
- Provide code reusability, reliability, modeling, reliability and flexibility.
- Provide robustness to the system.
- It provides roadmap for reusing objects in new software.

- The object oriented design process is consistent from analysis, through design to coding.

❖ Prepare use case diagram

- Use case model in UML (Unified Modeling Language) provides system behaviour.
- The use case model for any system consists of a set of "use cases".
- Use cases represent the different ways in which a system can be interacted with the users.

> ☛ *It is a representation of a user's interaction with the system that shows the relationship between the users and different use cases in which user is involved.*

- The purpose of a use case is to define the logical behaviour of the system without knowing the internal structure of it.
- Use case identifies the functional requirements of the system.
- Use case diagram describes "***who can do what in a system***".
- A use case typically represents a sequence of interactions between the user and the system.
- Use cases corresponding to the high-level functional requirements.
- For example → in ATM system, the system should have following use cases.
  - ➥ Check balance
  - ➥ Withdraw money
  - ➥ Deposit money
  - ➥ Transfer money
  - ➥ Print mini statement
  - ➥ Link aadhar card
  - ➥ Pin change etc.

➥ Components of use case diagram (Representation of use case diagram)

- Three main components along with relationships are used in use case diagram.

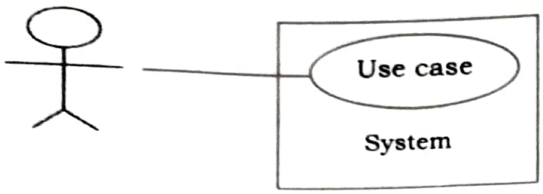
I. Use case

- Each use case is represented by an ellipse with the name of the use case written inside the ellipse, named by verb.
- All the use cases are enclosed with a rectangle representing system boundary. Rectangle contains the name of the system.
- It identifies, clarifies and analyzes the functional requirements of the system.

II. Actor

- An actor is anything outside the system that interacts with it, named by noun.
- Actors in the use case diagram are represented by using the stick person icon.
- An actor may be a person, machine or any external system.
- In use case diagram, actors are connected to use cases by drawing a simple line connected to it. Actor triggers use cases.

- *Each actor must be linked to at least one use case, while some use cases may not be linked to actors.*
- A simple figure showing the use cases and actor in the system.

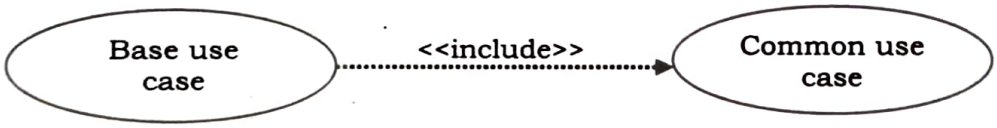


I. Relationship

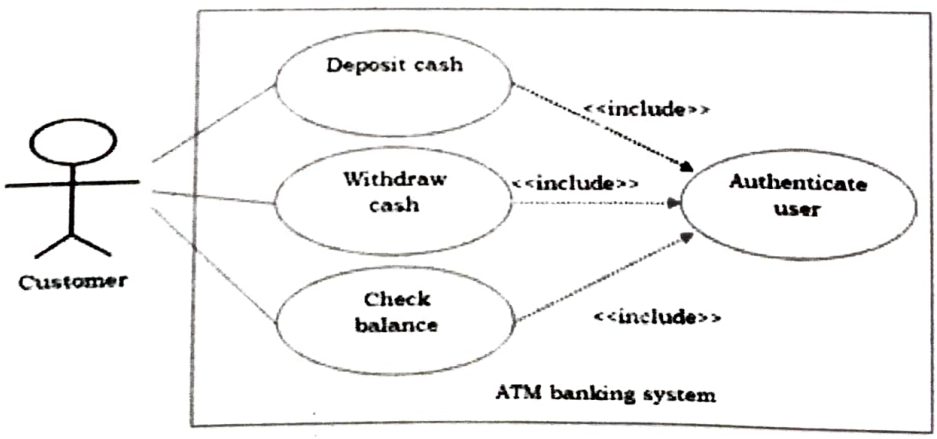
- Relationships are represented using 'a line' between an actor and a use case. It is also called communication relationship.
- An actor may have relationship with more than one use case and one use case may relate to more than one actor.

Different relationships in use case diagram are explained below:

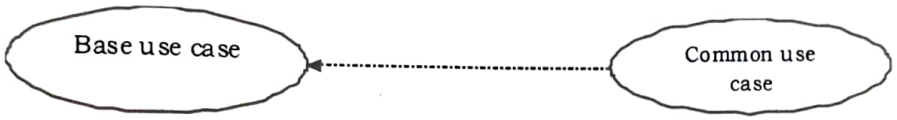
- **Association**
  - This relationship is the interface between an actor and a use case.
  - It is represented by joining a line from actor to use case.
- **Include relationship**
  - It involves one use case including the behaviour of another use case.
  - The "include" relationship occurs when a chunk of behaviour that is similar across a number of use cases.
  - It is represented using predefined stereotype <<include>>.
  - An include relationship is depicted with a directed arrow having a dotted line. The tip of arrowhead points to the child use case and the parent use case connected at the base of the arrow.



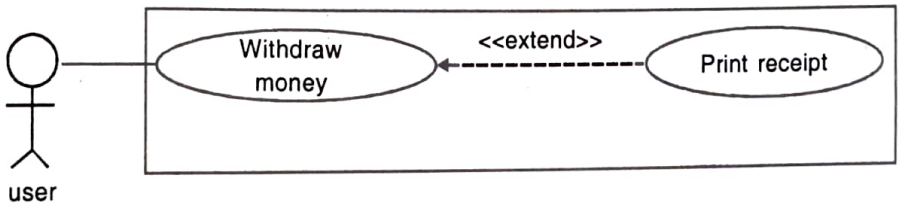
  - Example of include relationship is showing in below figure.



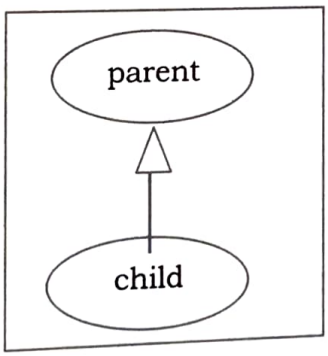
- **Extend relationship**
  - It allows you to show optional behaviour of the system. Extend is optional and supplementary.
  - This relationship among use cases is represented as a stereotype <<extend>>.
  - It is depicted with a directed arrow having a dotted line. The tip of arrowhead points to the base use case and the child use case is connected at the base of the arrow
  - Extend relationship exists when one use case calls another use case under certain condition (like: If – then condition).
  - Extend relationship is optional and supplementary.



Example of include relationship is showing in below figure.

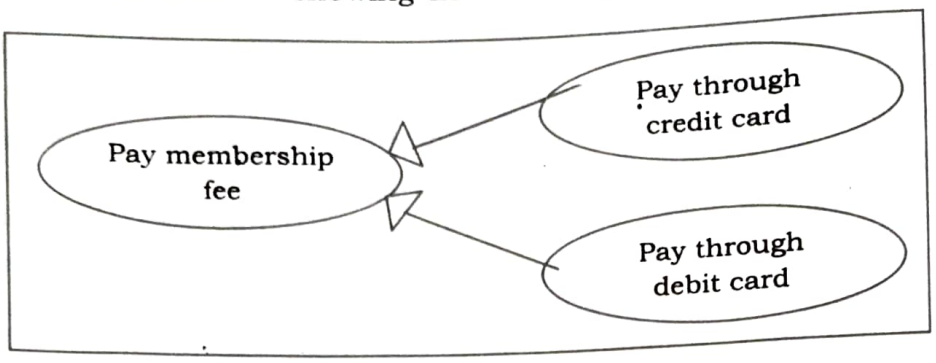


Generalization



- A generalization relationship is a parent-child relationship between use cases.
- Use case generalization can be used whenyou have one use case that is similar to another,but does slightly different.
- The child use case is an enhancement of the parent use case.
- Generalization is shown as a directed arrow with a triangle arrowhead. The child use case is connected at the base of the arrow. The tip of the arrow is connected to the parent use case.
- The child may override the behaviour of its parent.

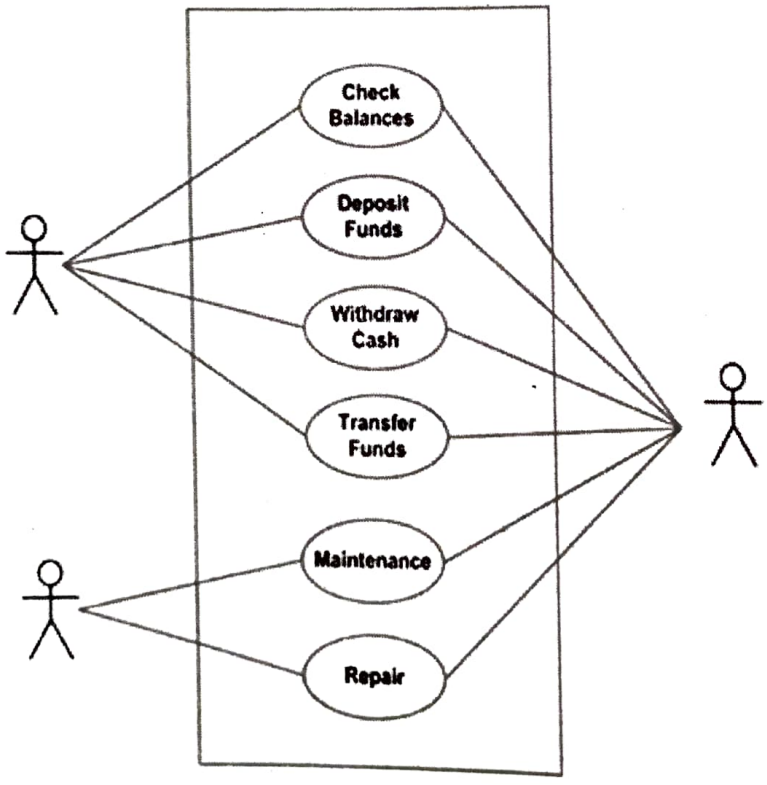
Example of generalization is showing in below figure.



➥ **Use case guidelines**

- Identify all different users. Give suitable names.
- For each user, identify tasks. These tasks will be the use cases.
- Use case name should be user perspective.
- Show relationships and dependencies.

Example → Use case diagram write bank ATM



➥ **Advantages of use case diagram**

- It is easy to understand and draw.
- It is used to capture the functional requirements of the system.
- It is used as basis for scheduling effort.
- It is used to verify whether all the requirements are captured.

Disadvantages of use case diagram

- Use case diagrams are not fully object oriented.
- It does not provide any guideline when to stop.

Applications of use case diagram

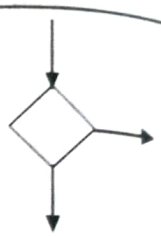
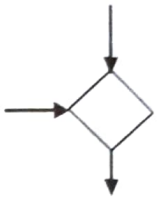
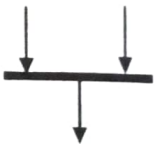
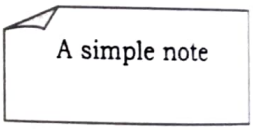
- Requirement analysis
- High level design
- Reverse engineering
- Forward engineering

Developing an Activity diagram

- An activity diagram falls under the category of behavioural diagram in UML.
- An activity diagram is a UML diagram that is used to model a process. It models the actions (behaviour) performed by the system components, the order in which the actions take place and conditions related to actions.
- Activity Diagrams consist of activities, states and transitions between activities and states.
- It describes how the events in a single use case relate to one another.
- The aim of activity diagram is to record the flow of control from one activity to another of each actor and to show interaction between them.
- It mainly represents series of actions and flow of control of a system.
- It focuses on the how of activities involved in a single process. Also shows how activities depend on one another.
- Activity diagrams represent workflows in a graphical way.
- Activity diagrams are similar to procedural flow charts. The difference is that activity diagram support parallel activities.
- An activity is a state with an internal action and one or more outgoing transitions.
- An interesting feature of the activity diagrams is the **swimlanes**. It enables you to group activities based on who is performing them. So, swimlanes make group of activities based on actors.

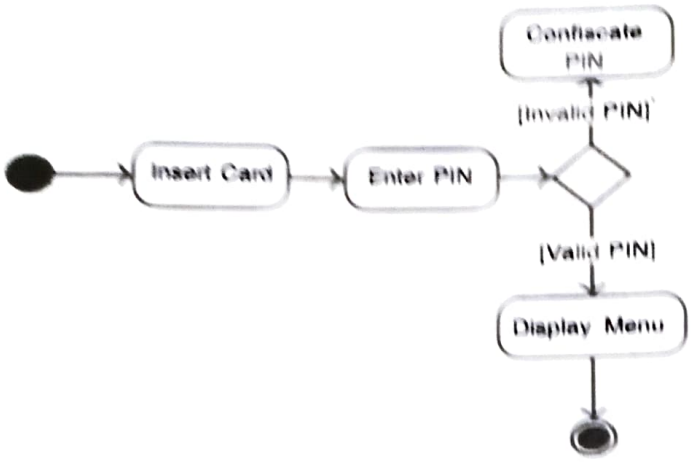
Elements (components) of an activity diagram :

Elements or Components and its description	Symbol
Activity - It represents a particular action taken in the flow of control. - It is denoted by a rectangle with rounded edges. And labelled inside it describing corresponding activity. - There are two special type of activity nodes: 1. **Initial activity (OR Start activity)** - This shows the starting point or first activity of the flow. - It is denoted by a solid circle. 2. **Final activity (OR End activity)** - The end of the activity diagram shown by a bull's eye symbol. It represents the end point of all activities.	An Activity  

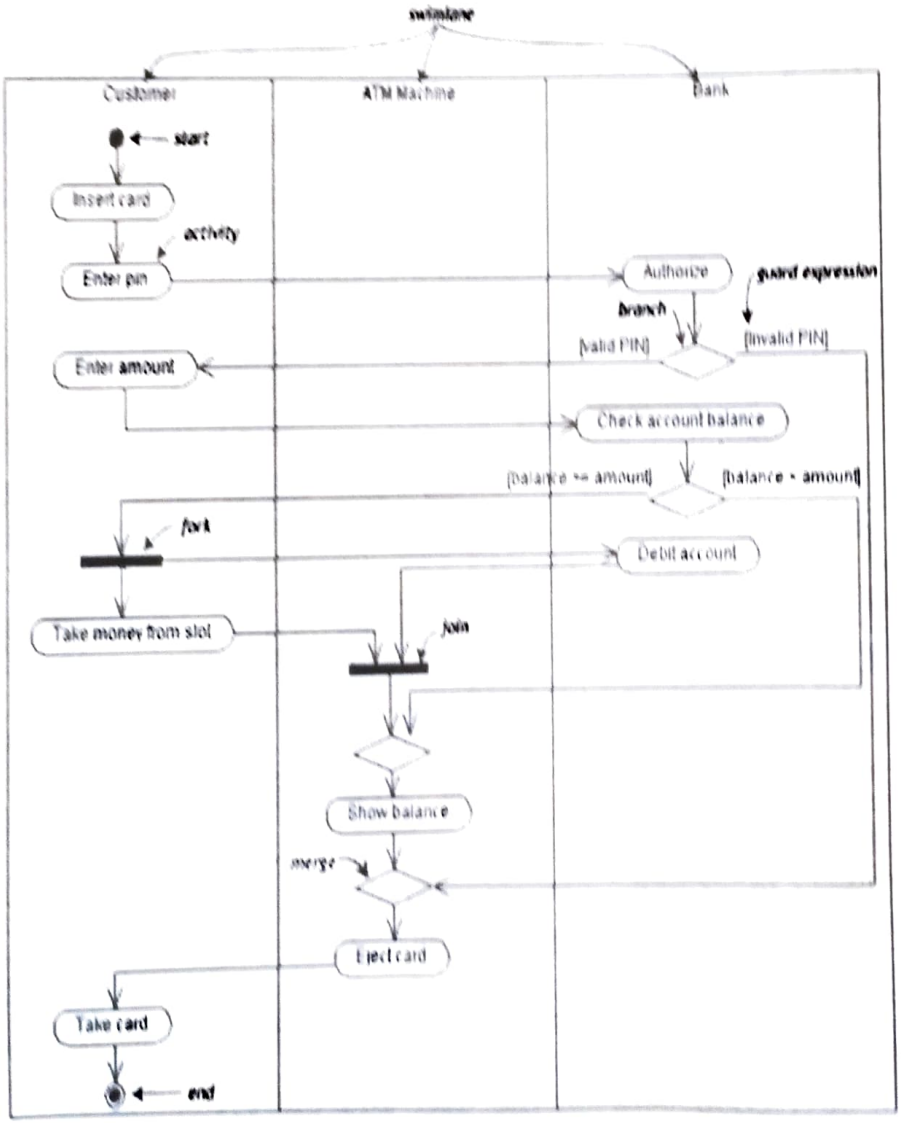
Flow or Transition - A flow (also termed as edge or transition) is represented with a directed arrow. - This is used to show transfer of control from one activity to another.	
Decision (Branch) - A decision node represented with a diamond. - It is a branch where single transition (flow) enters and several outgoing transitions.	
Merge - This is represented with a diamond shape with two or more input transitions and a single output transition.	
Fork - Fork is a point where parallel activities begin. - Fork is denoted by black bar with one incoming transition and several outgoing transitions. - When the incoming transition is triggered, all the outgoing transitions are taken into parallel.	
Join - Join is denoted by a black bar with multiple incoming transitions and single outgoing transition. - It represents the synchronization of all concurrent activities.	
Note - UML allows attaching a note to different components of diagram to present some textual information. - It could be some comments or may be some constraints. - A note generally attached to a decision point to indicate the branching criteria. - It is denoted by a rectangle with cut a side.	
Partition or Swimlanes - Different components of an activity diagram can be logically grouped into different areas, called partition or swimlanes. - They often correspond to different users or different units of organization. - It is denoted by drawing vertical parallel lines. - Partitions in an activity diagram are not mandatory.	
Guard conditions - Guard conditions control transition from alternative transitions based on condition. - These are represented by square brackets.	[ ]

Typical symbols used in activity diagram

A simple example activity diagram (ATM system)



Another example showing swimlanes



Advantages of activity diagram

- Activity diagrams can be very useful to understand complex processing activities.
- Different activities are grouped together based on actor. That is represented by *swimlanes*.
- It can be useful for analyzing a use case and understating workflow of system.
- Activity Diagrams are good for describing synchronization and concurrency between activities.
- Partitioning can be helpful in investigating responsibilities for interactions and associations between objects and actors.

Disadvantages of activity diagram

- The activity diagram does not provide message part. Means do not show any message flow from one activity to another.
- It can't describe how objects collaborate.
- It takes time to implement.
- Complex conditional logics (like Truth table) can't be represented by activity diagram.
- There are lot many symbols compare to other UML diagrams, so sometimes make it confusing to the developer.

When to use Activity diagram (Application of activity diagram)

- Mainly activity diagram used to describe the parallel behaviour of the system. It makes a great tool for workflow modeling.
- It is also used in multithreaded programming application.

Difference between flowchart and activity diagram

Flow chart	Activity diagram
→ It is limited for sequential access.	→ It is used for parallel and concurrent processing.
→ It is used for flow of control through an algorithm, not used for object oriented procedure.	→ It is usually used for object oriented systems.
→ Concept of swimlanes is not there in it.	→ It has the functionality of swimlanes.
→ It has limited functionalities compare to activity diagram.	→ It has more functionality.

2.1 User interface design

Basic concept of user interface

- User interface (UI) is the features of a computer system or device which allows the user to interact with it.
- A user interface, also sometimes called a human-computer interface, comprises both hardware and software components.
- It handles the interaction between the user and the system.

- It is a front end application to which user interacting to use the software. UI includes both input and output devices
- Example of user interface are → computer system, mouse, display screen, application program, operating system etc.
- UI mainly used to control the system or device to which users are interacting.

Characteristics of good user interface (Qualities of good UI)

UI is the most common way in interaction of human-computer. So UI should be designed in such a way that the users can accept it and use it.

Some important characteristics of good UI are:

- *Clear* : the whole purpose of UI should be clear to users.
- *Simple* : UI design should be simple to use. Less number of mouse clicks or keystrokes required to complete the task.
- *Consistent* : it provides consistency throughout the operation.
- *Prevention* : a good UI should prevents users from performing in-appropriate tasks.
- *Concise* : it should save the valuable time of users by keeping the things concise.
- *Proper responsive* : it should be responsive to the user actions and also give feedback to the users about what is happening.
- *Attractive* : a good UI should be attractive enough as users like to interact with them.
- *Efficient* : UI needs to be efficient to figure out what exactly the user is trying to achieve.
- *Reliable* : a UI should be reliable so that user can use it for a long time with trust and interest.
- *Maintainable* : a UI should have the capacity for and changes to be integrated without causing a conflict of interest.
- *User guidance and online assistance* : a good user interface s one which additionally offers assist to its users at the time of help at some point of time.

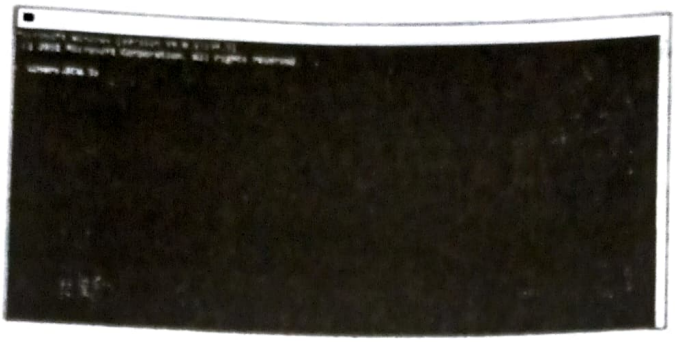
Types of User Interface

- **Different types of user interface are :**

✓ Graphical user interface (GUI)
✓ Command line interface (CLI)
✓ Menu driven user interface
✓ Direct manipulation interface
✓ Touch user interface
✓ Form based user interface
✓ Voice user interface (VUI)

(*Note:* most of the books and websites classify UI in three types: GUI, CLI and menu based UI). Here we will discuss only those types which are in syllabus.

➥ Command based (OR Command line) user interface (CLI)



- A command is a text-based reference to set of instructions, which are expected to be executed by the system.
- Command-line interfaces are also called console user interfaces or character user interfaces.
- CLI is a text-based user interface (UI) used to run programs, manage computer files and interact with the computer.
- CLI provides a command prompt, the place where the user types the command and feeds to the system. The user needs to remember the syntax of command and its use.

○ Elements of CLI :

- Command prompt: it is a text based editor in which user is working.
- Cursor: It is a small horizontal or vertical line to represent position of character while typing.
- Command: it is an executable instruction.

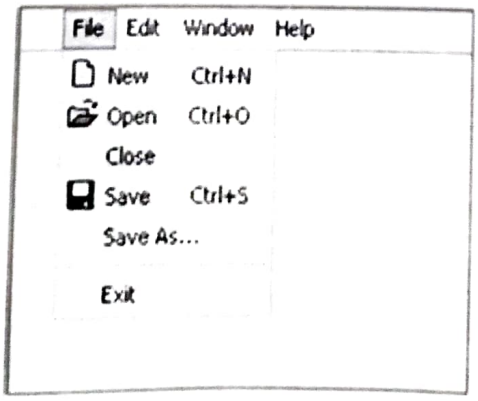
○ Advantages of CLI

- Simple to use.
- Minimal memory usage.
- Don't require high computing facility computer.

○ Disadvantages of CLI

- Difficult to learn command language.
- Minimal error information.
- Complex for new users.

➥ Menu based (OR Menu driven) user interface



- A menu-driven interface is simply an easier way of navigating the devices and programs to interact with them.
- It consists of a series of screens, or "menus," that allow users to make choices about what to do next by clicking / tapping on the given options.
- It is preferred for their simplicity and user-friendly properties.
- ATM is best example of this type of interface where user can select particular menu for his banking operation.

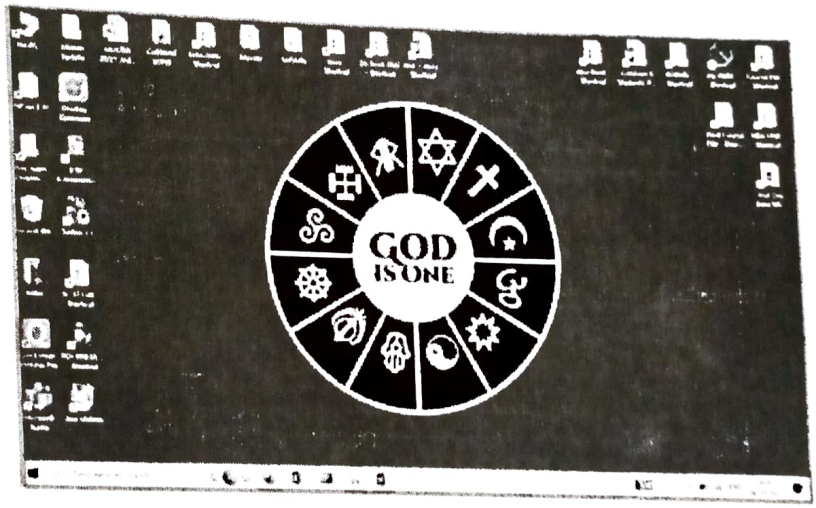
Advantages of menu based UI

- Easy to use, even to novice users.
- No need to remember a list of commands.
- Self-explanatory menu options should be there.
- More controls over user interactions.

Disadvantages of menu based UI

- Limited menu options.
- Sub-menu might be difficult to find.
- Requires necessary actions for a simple task.

Graphical user interface (GUI)



- It is the type of interface with which the majority of people are the most familiar.
- You interact with these interfaces by using a mouse, tack pad, or other peripheral to point and click on graphics or icons.
- It is developed by Xerox, the GUI was popularized by the Apple Macintosh in the 1980s.
- A graphics-based operating system interface that uses icons, menus and a mouse (to click on the icon or pull down the menus) to manage interaction with the system.
- GUI can be combination of both hardware and software.
- Microsoft windows is one of the most suitable example for GUI.
- A GUI includes GUI objects, like icons, cursors, buttons, tabs, windows and many more. In above figure we can see some GUI objects like – start button, icons, date and time, task bar, notification area, battery status etc.
- Typically, GUI is more resource consuming than that of CLI.

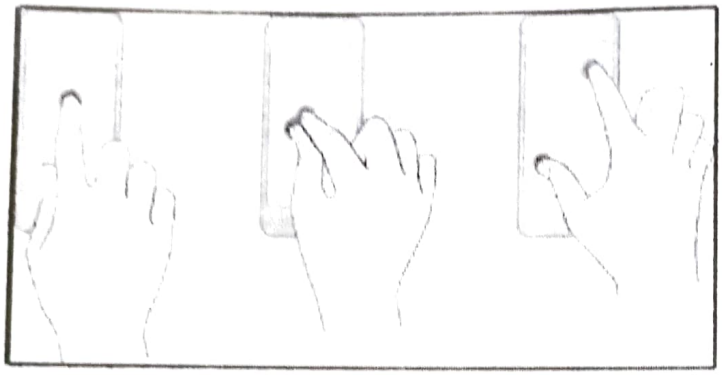
Advantages of GUI

- Easier to learn compare with CLI and menu based UI.
- Attractive and ease of use with faster manipulation.
- Self-descriptive and immediate feedback.
- Suitable for non-technical users.
- Enables usage of multiple input devices.
- The complexity of actions is hidden from the users.

Disadvantages of GUI

- Required power and high amount of memory resources.
- Works slower than CLI.
- It's comparably difficult for the developers to create an intuitive GUI.

Direct manipulation user interface



- Direct manipulation (DM) is an interaction style in which users act on displayed objects of interest using physical, incremental, reversible actions whose effects are immediately visible on the screen.
- In more simple term we can say that: a direct manipulation allows a user to directly act on a set of objects (instruments) in the interface.
- Users get immediate feedback about their actions and the outcome of these actions.
- It permits novice users access to powerful facilities without the burden of learning to use a complex syntax and lengthy list of commands.
- People are playing games on mobile is the best example of this interface.
- Some other examples: dialing a phone number by pushing number on the screen, dragging a document to recycle bin etc.

Advantages of direct manipulation interface

- Users can feel direct involvement with the work.
- Continuous visibility of objects and actions.
- Errors can be avoided more easily.
- Experts can work extremely rapidly to carry out a wide range of tasks, and novice users can learn basic functionality quickly.

Rapid, reversible and incremental actions.

Disadvantages of direct manipulation

- DM is slow in some sense. If the user needs to perform a large number of actions, on many objects, using direct manipulation takes a lot longer than a command-line interface.
- Repetitive tasks are not well supported.
- Some gestures can be more error-prone than typing

ASSIGNMENT

MCQs

1. Among the following which one is of bottom up approach ?
   (a) Data flow diagram (b) ER Diagram
   (c) Structure chart **(d) Class diagram**
2. The full form of DFD is ..... .
   (a) Data Flow diagram (b) Data Function Diagram
   (c) Detailed Flow Diagram (d) Detailed Function Diagram
3. Which among the following is better or any software development.
   (a) Low cohesion low coupling (b) Low cohesion high coupling
   (c) High cohesion high coupling **(d) High cohesion low coupling**
4. Highest cohesion is ..... .
   (a) Coincidental **(b) Functional** (c) Logical (d) Procedural
5. Highest coupling is ..... .
   (a) Data (b) Control **(c) Content** (d) Common
6. Which cohesion is best among the following ?
   (a) Communicational (b) Sequential
   (c) Logical **(d) Functional**
7. Level 0 DFD is also known as ..... .
   (a) activity diagram (b) use case diagram
   (c) context diagram (d) class diagram
8. Which of the following features that we can see in activity diagram but not in flow chart ?
   (a) swimlanes (b) activity (c) condition (d) flow
9. At the time of designing activity diagram, if we want to group the activity logically, which of the following features is used ?
   (a) fork (b) activity
   (c) guard condition **(d) swimlanes**
10. The full form of GUI is :
    (a) Graphical User Input (b) General User Input
    (c) **Graphical User Interface** (d) General User Interface

True / False

1. Low cohesion and high coupling is better for any software development. **Answer : False**
2. Flow chart can be used for parallel processes. **Answer : False**
3. In temporal cohesion, all the elements of a module are executed in same time span. **Answer : True**
4. GUI is faster compare with CLI. **Answer : False**

Short Questions

1. Give the full form of :
   - DD - ERD - OOD - CLI
   - GUI - DFD - UML - GUI
2. Intra dependency of module is ..... and inter dependency of module is .....
   Answer : cohesion,, coupling
3. List out classification of design methodologies.
4. Define cohesion and coupling.
5. Justify: why high cohesion and low coupling is good for any software design.
6. Draw symbols used in DFD.
7. ..... type of data dictionary maintained automatically.
   Answer : active data dictionary
8. When to use use-case diagram?
9. List advantages of object oriented design.

Descriptive Questions

1. Explain classification of design activities.
2. Explain classification of design methodology.
3. Write short a note on function oriented design.
4. Write short a note on cohesion. **(OR** Explain cohesion with its classification.)
5. Write short a note on coupling. **(OR** Explain coupling with its classification.)
6. Explain primitive symbols used in DFD.
7. Explain advantages and disadvantages of DFD.
8. Write a short note on data dictionary.
9. Explain DFD with example.
10. Explain symbols used in activity diagram. **(OR** Writeashort note on activity diagram.)
11. What is the significance of activity diagram in SE ? Explain in brief.
12. Explain object oriented design.
13. Draw context diagram for library management system.
14. Draw a use case diagram for online shopping web application.
15. Explain GUI with its advantages and disadvantages.
16. Write a short note on CLI.

* * *