SOFTWARE ENGINEERING UNIT: 3

Function Oriented Software Design: Overview of SA/SD Methodology, Structured Analysis, Developing the DFD model of the system, Structured Design, Design Review, Overview of Object oriented Design

PART 1: FUNCTION ORIENTED SOFTWARE DESIGN

OVERVIEW OF SA/SD METHODOLOGY

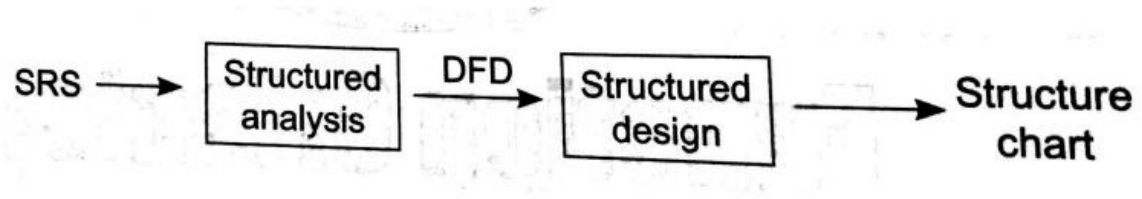
Function is the sub program of software and all the functions are organized using Function oriented design methodology.

The function oriented design methodology is the process oriented approach for developing the software solution. The design involves the structured oriented mechanism for Analysis and Design.

The following are two activities –

- Structure Analysis (SA)
- Structure Design (SD)

The working of these two activities in function oriented design is shown below. Here the SRS document is used as input for structured analysis to produce the data flow diagram. The DFD is used to perform structured design and produces the structure chart.



The structured analysis is carried out to understand the problem where as structured design is performed to design the solution. System analyst begins with SRS document and design DFD to understand the dataflow in the system. Final DFD is used to carry out the structured design. It follows top-down decomposition approach to design a hierarchical structure chart.

STRUCTURED ANALYSIS

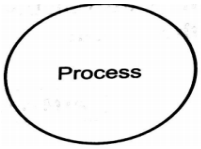
Structured analysis is also referred to as *process modeling* or *data flow modeling.* It is useful to understand the working process of the system. It represents the behavior of the system in an abstract representation. It follows a top-down functional decomposition process.

Structured analysis use a graphical tool called data flow diagrams (DFD), which represents the system behavior.

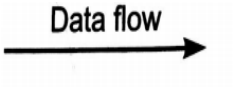
❖ Data Flow Diagram (DFD)

A DFD is a graphical tool that describes the flow of data through a system and the functions performed by the system. It is also known as Process model or Information flow model. DFD has four different symbols like – Process, Data Flow, Data Store, Actor are explained below.

- **PROCESS –** A process is represented by a circle and denotes transformations of input data to produce the output data. This output data is used as input data for another sub process of the system.

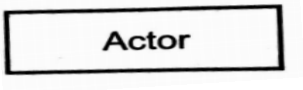


- **DATA FLOW -** Data flow are data in motion. It is represented by arrows, connecting one data transformation to another. These may be discrete or composite data.



- **DATA STORE –** Data store is a data at rest. Incoming arrows indicate the updating of data store while outgoing arrows indicate the retrieval of data from the data store. It represents by an open ended horizontal rectangle or parallel lines.

- **ACTOR –** It is the external entity that represents the source or sink (destination). It is represented by a rectangle. It represents the people, organization, or another system.

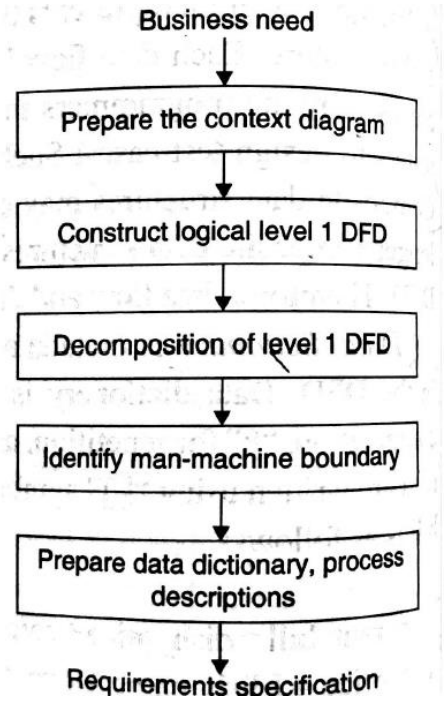


❖ Data Dictionary

The content that are not described in the DFD are described in data dictionary. **Data Dictionary** is a **metadata** that describes the composite data structures defined in DFD. It defines the data store and relevant meaning. A physical data dictionary for data elements which flow between processes, between entities, and between processes and entities may be included. This would also include descriptions of data elements that flow external to the data stores.

❖ Structured Analysis Method

Structured Analysis uses data flow diagrams and data dictionary for requirement analysis. This approach begins with studying the existing system to understand the working of system to design the context diagram. The structured analysis approach is shown below.



❖ **Pros and Cons of Structured Analysis**

- A data flow based structured analysis is easy to present the customer problems in a pictorial form that can easily be converted into structured design. The DFD based approach require the technical expertise.
- It is difficult to understand to understand the final DFD and also doesn't reveal the sequence in which processes are performed in the system. This methodology becomes a time consuming job to manage such voluminous data and produce several levels of DFD.

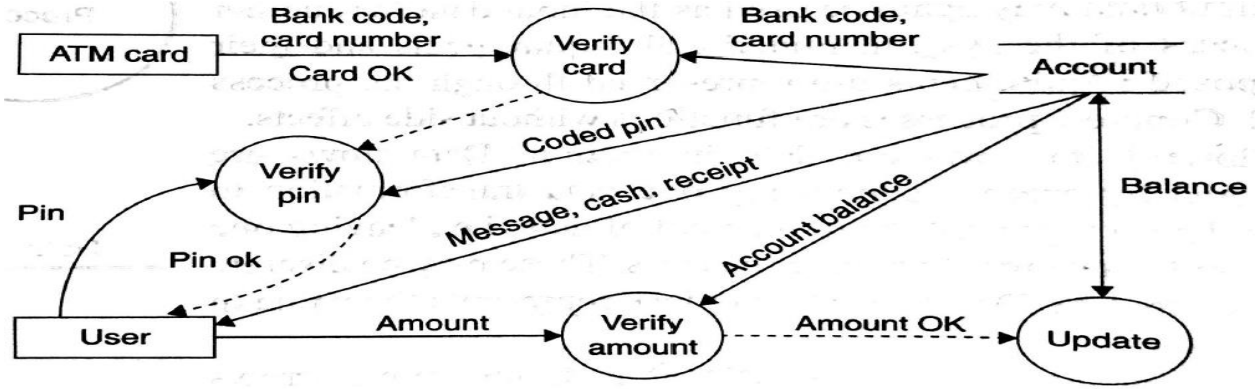
DEVELOPING THE DFD MODEL OF THE SYSTEM

A data flow diagram (DFD) shows the way information flows through a process or system. It includes data inputs and outputs, data stores, and the various sub processes the data moves through. DFDs are built using standardized symbols and notation to describe various entities and their relationships.

They can be used to analyze an existing system or model a new one. Like all the best diagrams and charts, a DFD can often visually "say" things that would be hard to explain in words, and they work for both technical and nontechnical audiences, from developer to CEO. That's why DFDs remain so popular after all these years. While they work well for data flow software and systems, they are less applicable nowadays to visualizing interactive, real-time or database-oriented software or systems.

Example: Here we consider Cash Withdrawal from ATM machine as an example of DFD. It uses two actors, user and ATM card, four processes: verify card, verify pin, verify amount and update balance and a data store ie., amount.

The construction of DFD starts with high level functionality og the system, which incorporates external inputs and outputs.



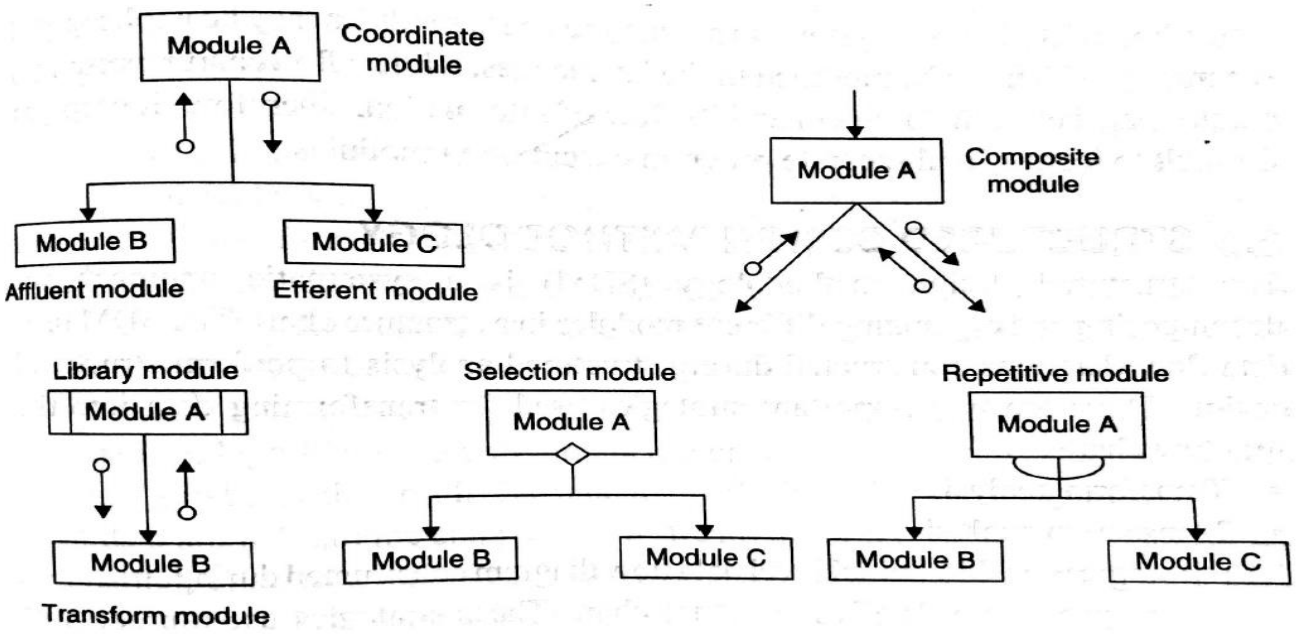
STRUCTURED DESIGN

The basic approach of structured design is to transform the data flow diagrams of structured analysis into structure charts. It is represented through the structured chart and it follows the transform analysis and transaction analysis to produce the design of the system.

Structured Chart: It is a graphical representation of procedural programs in the structured design methodology. It represents the modules of a system in a hierarchical fashion. Hence, the structure chart representation can be easily implemented using some programming language. Since the main focus in a structure chart representation is on the module structure of the software and the interactions among different modules, the procedural aspects (e.g. how a particular functionality is achieved) are not represented.

The building blocks in the software are represented through modules and these are linked together to produce the final design. The following symbols are used to represent the structure chart and these symbols are illustrated in Figure 5.8.

1. **Affluent module:** The affluent module, also known as the input module, receives information from a subordinate module and passes on to a superordinate module. This module is used to collect the input data.
2. **Efferent module:** The efferent module, also known as the output module, passes information from a superordinate to a subordinate module. This module is used to produce intermediate or final outcomes.
3. **Transform module:** The transform module performs data transformation. It receives data, performs some operations, and represents them into another form.
4. **Coordinate module:** The coordinate module manages the transformations communication between subordinate modules.
5. **Composite module:** The composite module can perform functions at more than one module.
6. **Selection:** The selection of a module is represented through a diamond box. The control couple decides which subordinate module is to be invoked for further operation.
7. **Repetition:** The repetition of a module is represented by a looping arrow around the modules to be iterated in the program.
8. **Library module:** The library module is represented by a rectangle with double edges. The frequently called modules are iterated in the design.



DETAILED DESIGN

The detailed design concentrates on specifying the procedural descriptions, data structures representations, interfaces between data structures and functions and packaging of the software product. The basis of the detailed design is the blueprint, for which algorithms details and data structures are provided. There are various detailed design tools such as algorithm, pseudo code, Program Design Language, structured English, and so on.

Program Design Language:

The Program Design Language (PDL) is a software design tool to which is used to produce structured design in a top-down manner. PDL statements can be implemented in programming language without any efforts. The most common constructs used in PDL are IF-THEN-ELSE, DO, DO-WHILE, DO-UNTIL, DO-FOR-EXCEPT, and CASE-OF. PDL programs can be used to produce source codes of programming languages through PDL procedures.

```
SUM (ARRAY A, N)
     Initialize total to 0
     DO FOR N input
          read a number into A
     ENDDO
     DO WHILE there are numbers in A
          add number to total
     print (total)
END
```

Algorithmic Design:

An Algorithm is a step-by-step process of problem solving. It has two parts namely, problem definition and design of algorithm. It is written in a formal statements or pseudo code. The design of an algorithm is

done through step-wise refinement, in which algorithm is broken down into smaller parts so that it can be solved in a convenient and manageable manner. Below is an algorithm for binary search -

```
Algorithm Binary_Search (A,N,X). This algorithm searches the
element X in an array A of N elements arranged in an ascending
order. The variables Low, Mid, and High represent the lower,
middle, and upper limits of the search interval, respectively.
This algorithm prints the middle element in case of successful
search, otherwise it prints unsuccessful search.
1.   [Initialize]
     Low = 1, High = N
2.   [Perform search]
     Repeat thru step 4 while Low<=High
3.   [Find the middle element]
     Mid = [(Low + High)/2]
4.   [Compare]
     If X < A[Mid]
     then High= Mid-1
     else If X > A[Mid]
        then Low = Mid + 1
        else Write ('Successful search')
        Write (Mid)
5.   [Unsuccessful search]
     Write (`Unsuccessful search')
     Write (`Element not found')
```

DESIGN REVIEW

The outcome of design should be verified, which is to be used in subsequent stages of software development. Design review and verification ensures that all the requirements have been incorporated in the design. A good design can be easily be modified and maintained in the future. A systematic design can be easily converted into the programming languages.

There are two most common approaches used for design verification and these are ***design reviews*** and ***design inspections***. The Design reviews are conducted at the end of software design. A review meeting is called to discuss whether it satisfies all the conditions and constraints laid down in SRS document. Design inspection is performed by expert team members who are experienced in the functional area. A checklist of items is provided to the team members to inspect the design.

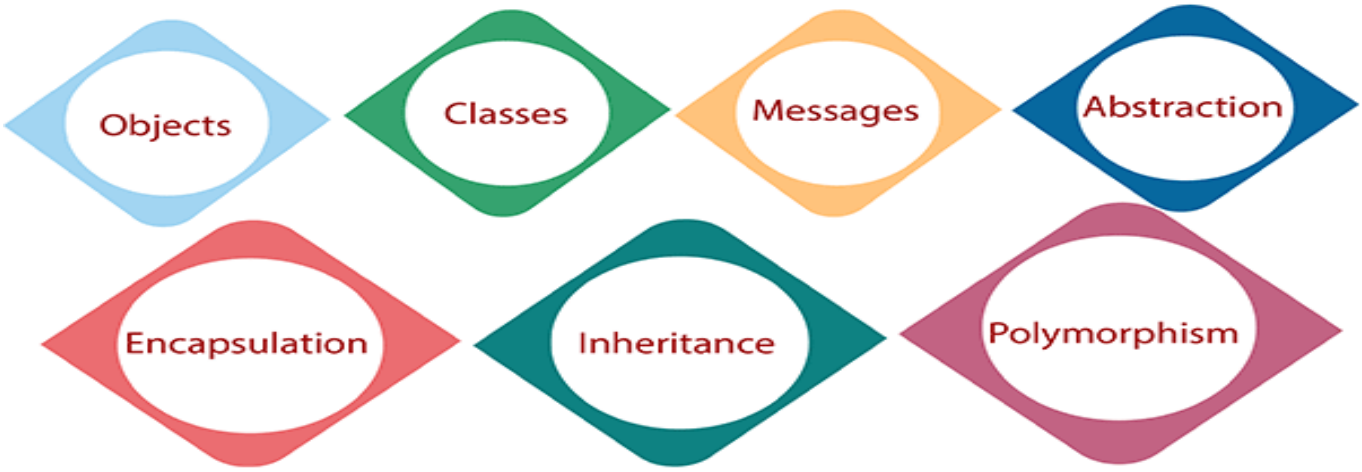
Design is verified with the help of checklist and a detailed report is prepared and given to the project coordinator.

OVERVIEW OF OBJECT ORIENTED DESIGN

Object oriented analysis and design (OOAD) is a combined approach to problem analysis and designing the solution. OOAD has two important aspects like Object oriented analysis (OOA) and object oriented design (OOD).

OBJECT ORIENTED CONCEPTS:

In the object-oriented design method, the system is viewed as a collection of objects (i.e., entities). The state is distributed among the objects, and each object handles its state data.



- **Objects:** All entities involved in the solution design are known as objects. For example, person, banks, company, and users are considered as objects. Every entity has some attributes associated with it and has some methods to perform on the attributes.
- **Classes:** A class is a generalized description of an object. An object is an instance of a class. A class defines all the attributes, which an object can have and methods, which represents the functionality of the object.
- **Messages:** Objects communicate by message passing. Messages consist of the integrity of the target object, the name of the requested operation, and any other action needed to perform the function. Messages are often implemented as procedure or function calls.
- **Abstraction** In object-oriented design, complexity is handled using abstraction. Abstraction is the removal of the irrelevant and the amplification of the essentials.
- **Encapsulation:** Encapsulation is also called an information hiding concept. The data and operations are linked to a single unit. Encapsulation not only bundles essential information of an object together but also restricts access to the data and methods from the outside world.
- **Inheritance:** OOD allows similar classes to stack up in a hierarchical manner where the lower or sub-classes can import, implement, and re-use allowed variables and functions from their immediate superclasses. This property of OOD is called an inheritance. This makes it easier to define a specific class and to create generalized classes from specific ones.
- **Polymorphism:** OOD languages provide a mechanism where methods performing similar tasks but vary in arguments, can be assigned the same name. This is known as polymorphism, which allows a single interface is performing functions for different types. Depending upon how the service is invoked, the respective portion of the code gets executed.