Module 3

What is Structured Analysis?

Structured Analysis is a development method that allows the analyst to understand the system and its activities in a logical way.

It is a systematic approach, which uses graphical tools that analyze and refine the objectives of an existing system and develop a new system specification which can be easily understandable by user.

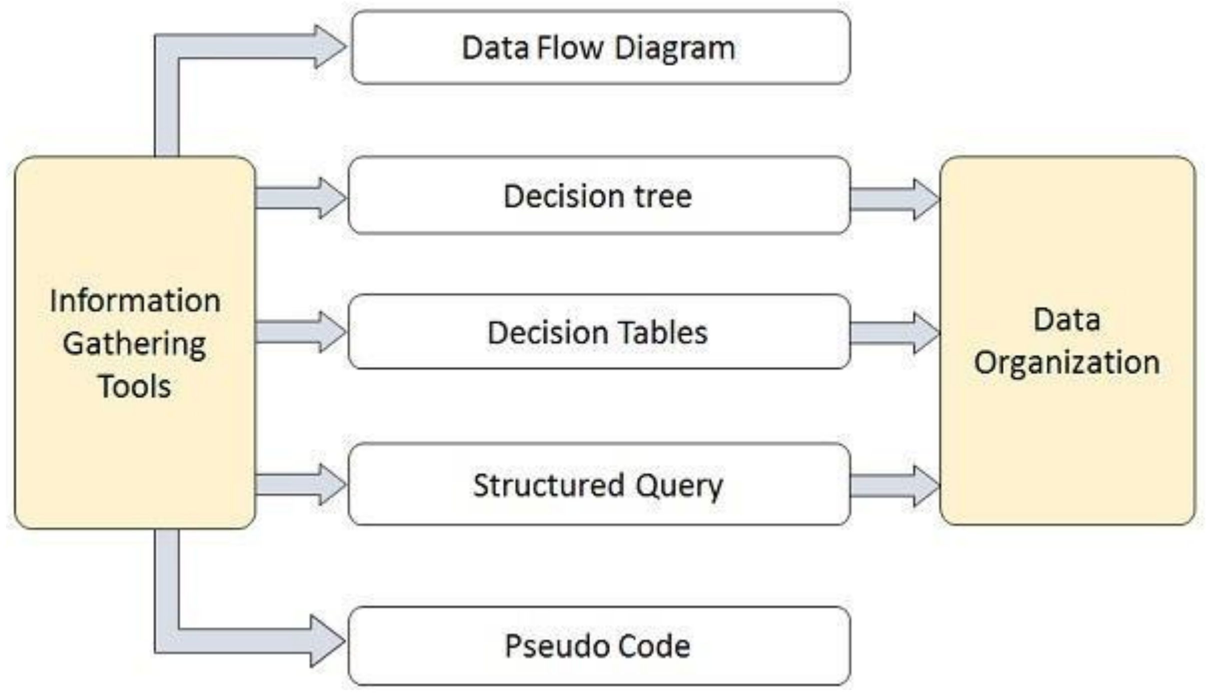
It has following attributes:

- It is graphic which specifies the presentation of application.
- It divides the processes so that it gives a clear picture of system flow.
- It is logical rather than physical i.e., the elements of system do not depend on vendor or hardware.
- It is an approach that works from high-level overviews to lower-level details.

Structured Analysis Tools

During Structured Analysis, various tools and techniques are used for system development. They are:

- Data Flow Diagrams
- Data Dictionary
- Decision Trees
- Decision Tables
- Structured English
- Pseudocode



Data Flow Diagrams (DFD) or Bubble Chart

It is a technique developed by Larry Constantine to express the requirements of system in a graphical form.

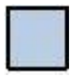
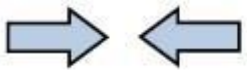
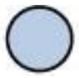
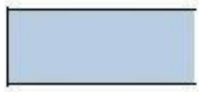
- It shows the flow of data between various functions of system and specifies how the current system is implemented.
- It is an initial stage of design phase that functionally divides the requirement specifications down to the lowest level of detail.
- Its graphical nature makes it a good communication tool between user and analyst or analyst and system designer.
- It gives an overview of what data a system processes, what transformations are performed, what data are stored, what results are produced and where they flow.

Basic Elements of DFD

DFD is easy to understand and quite effective when the required design is not clear and the user wants a notational language for communication. However, it requires a large number of iterations for obtaining the most accurate and complete solution.

The following table shows the symbols used in designing a DFD and their significance −

Symbol Name	Symbol	Meaning

Square		Source or Destination of Data
Arrow		Data flow
Circle		Process transforming data flow
Open Rectangle		Data Store

Types of DFD

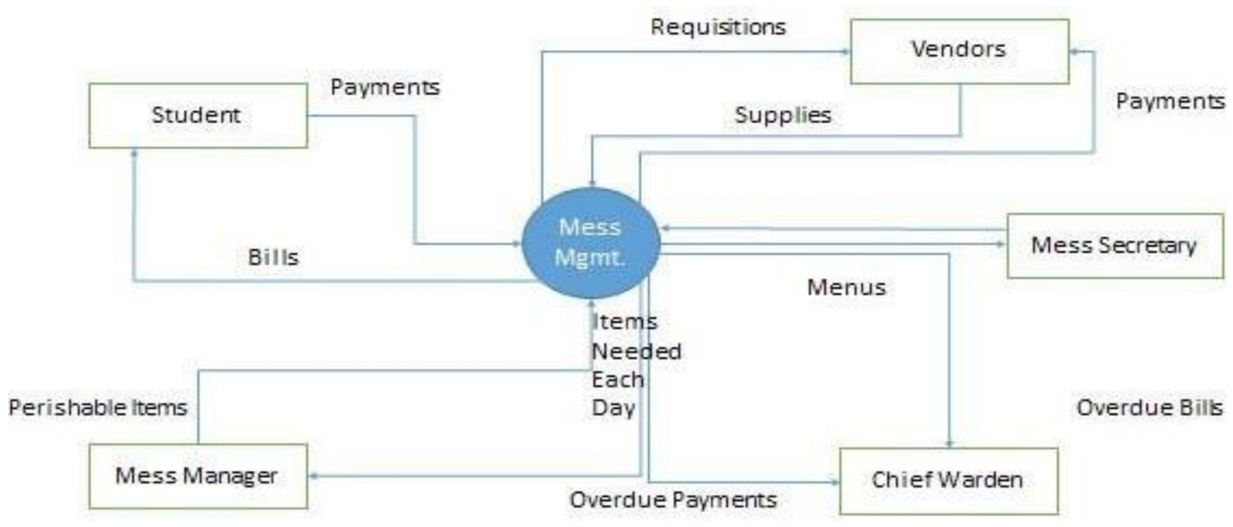
DFDs are of two types: Physical DFD and Logical DFD. The following table lists the points that differentiate a physical DFD from a logical DFD.

Physical DFD	Logical DFD
It is implementation dependent. It shows which functions are performed.	It is implementation independent. It focuses only on the flow of data between processes.
It provides low level details of hardware, software, files, and people.	It explains events of systems and data required by each event.
It depicts how the current system operates and how a system will be implemented.	It shows how business operates; not how the system can be implemented.

Context Diagram

A context diagram helps in understanding the entire system by one DFD which gives the overview of a system. It starts with mentioning major processes with little details and then goes onto giving more details of the processes with the top-down approach.

The context diagram of mess management is shown below.



Data Dictionary

A data dictionary is a structured repository of data elements in the system. It stores the descriptions of all DFD data elements that is, details and definitions of data flows, data stores, data stored in data stores, and the processes.

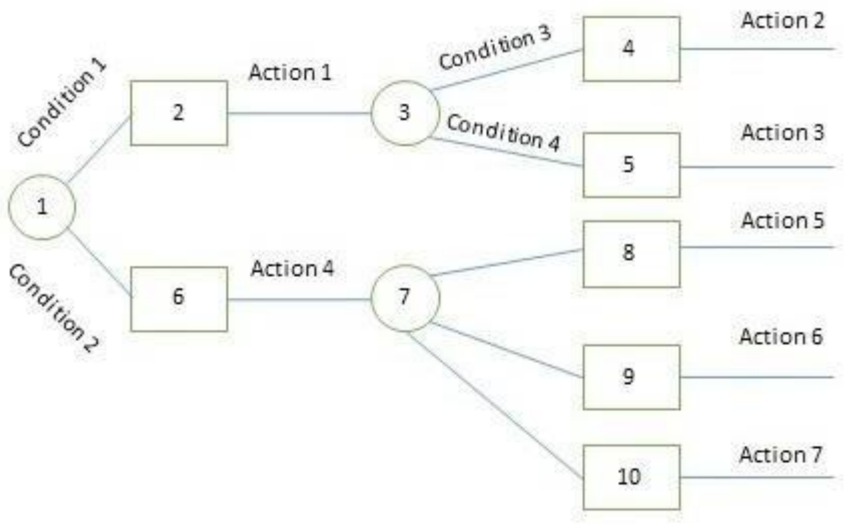
A data dictionary improves the communication between the analyst and the user. It plays an important role in building a database. Most DBMSs have a data dictionary as a standard feature. For example, refer the following table

Sr.No.	Data Name	Description	No. of Characters
1	ISBN	ISBN Number	10
2	TITLE	title	60
3	SUB	Book Subjects	80
4	ANAME	Author Name	15

Decision Trees

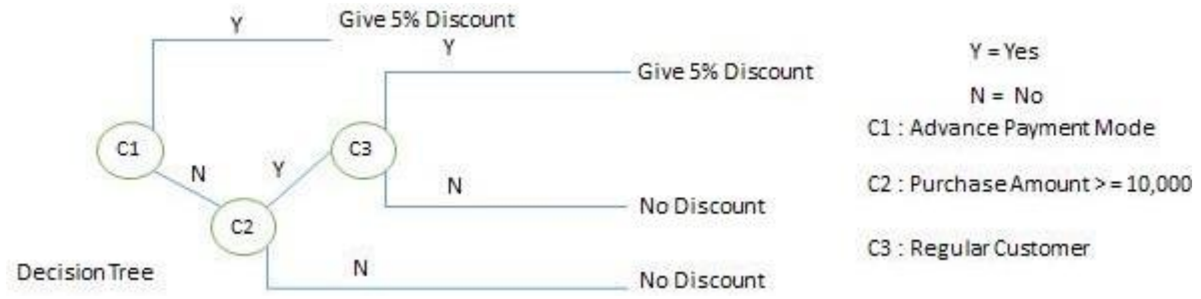
Decision trees are a method for defining complex relationships by describing decisions and avoiding the problems in communication. A decision tree is a diagram that shows alternative actions and conditions within horizontal tree framework. Thus, it depicts which conditions to consider first, second, and so on.

Decision trees depict the relationship of each condition and their permissible actions. A square node indicates an action and a circle indicates a condition. It forces analysts to consider the sequence of decisions and identifies the actual decision that must be made.



The major limitation of a decision tree is that it lacks information in its format to describe what other combinations of conditions you can take for testing. It is a single representation of the relationships between conditions and actions.

For example, refer the following decision tree −



Decision Tables

Decision tables are a method of describing the complex logical relationship in a precise manner which is easily understandable.

- It is useful in situations where the resulting actions depend on the occurrence of one or several combinations of independent conditions.
- It is a matrix containing row or columns for defining a problem and the actions.

Components of a Decision Table

- **Condition Stub** − It is in the upper left quadrant which lists all the condition to be checked.
- **Action Stub** − It is in the lower left quadrant which outlines all the action to be carried out to meet such condition.
- **Condition Entry** − It is in upper right quadrant which provides answers to questions asked in condition stub quadrant.
- **Action Entry** − It is in lower right quadrant which indicates the appropriate action resulting from the answers to the conditions in the condition entry quadrant.

The entries in decision table are given by Decision Rules which define the relationships between combinations of conditions and courses of action. In rules section,

- Y shows the existence of a condition.
- N represents the condition, which is not satisfied.
- A blank - against action states it is to be ignored.
- X (or a check mark will do) against action states it is to be carried out.

For example, refer the following table −

CONDITIONS	**Rule 1**	**Rule 2**	**Rule 3**	**Rule 4**
Advance payment made	Y	N	N	N
Purchase amount = Rs 10,000/-	-	Y	Y	N
Regular Customer	-	Y	N	-
ACTIONS				

Give 5% discount	X	X	-	-
Give no discount	-	-	X	X

Structured English

Structure English is derived from structured programming language which gives more understandable and precise description of process. It is based on procedural logic that uses construction and imperative sentences designed to perform operation for action.

- It is best used when sequences and loops in a program must be considered and the problem needs sequences of actions with decisions.
- It does not have strict syntax rule. It expresses all logic in terms of sequential decision structures and iterations.

For example, see the following sequence of actions −

```
if customer pays advance
  then
    Give 5% Discount
  else
    if purchase amount >=10,000
      then
        if  the customer is a regular customer
          then Give 5% Discount
        else  No Discount
      end if
    else No Discount
  end if
end if
```

Pseudocode

A pseudocode does not conform to any programming language and expresses logic in plain English.

- It may specify the physical programming logic without actual coding during and after the physical design.
- It is used in conjunction with structured programming.
- It replaces the flowcharts of a program.

Guidelines for Selecting Appropriate Tools

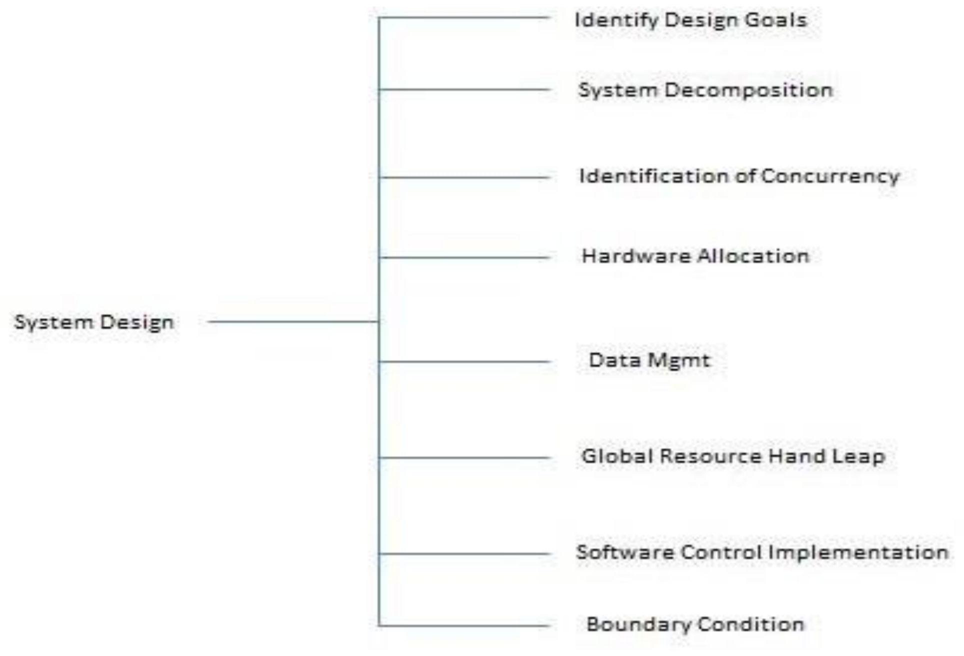
Use the following guidelines for selecting the most appropriate tool that would suit your requirements:

- Use DFD at high or low level analysis for providing good system documentations.
- Use data dictionary to simplify the structure for meeting the data requirement of the system.
- Use structured English if there are many loops and actions are complex.
- Use decision tables when there are a large number of conditions to check and logic is complex.
- Use decision trees when sequencing of conditions is important and if there are few conditions to be tested.

System design is the phase that bridges the gap between problem domain and the existing system in a manageable way. This phase focuses on the solution domain, i.e. "how to implement?"

It is the phase where the **SRS** document is converted into a format that can be implemented and decides how the system will operate.

In this phase, the complex activity of system development is divided into several smaller sub-activities, which coordinate with each other to achieve the main objective of system development.



Inputs to System Design

System design takes the following inputs:

- Statement of work
- Requirement determination plan
- Current situation analysis
- Proposed system requirements including a conceptual data model, modified DFDs, and Metadata (data about data).

Outputs for System Design

System design gives the following outputs:

- Infrastructure and organizational changes for the proposed system.
- A data schema, often a relational schema.
- Metadata to define the tables/files and columns/data-items.
- A function hierarchy diagram or web page map that graphically describes the program structure.
- Actual or pseudocode for each module in the program.
- A prototype for the proposed system.

Types of System Design

Logical Design

Logical design pertains to an abstract representation of the data flow, inputs, and outputs of the system. It describes the inputs (sources), outputs (destinations), databases (data stores), procedures (data flows) all in a format that meets the user requirements.

While preparing the logical design of a system, the system analyst specifies the user needs at level of detail that virtually determines the information flow into and out of the system and the required data sources. Data flow diagram, E-R diagram modeling are used.

Physical Design

Physical design relates to the actual input and output processes of the system. It focuses on how data is entered into a system, verified, processed, and displayed as output.

It produces the working system by defining the design specification that specifies exactly what the candidate system does. It is concerned with user interface design, process design, and data design.

It consists of the following steps:

- Specifying the input/output media, designing the database, and specifying backup procedures.
- Planning system implementation.

- Devising a test and implementation plan, and specifying any new hardware and software.
- Updating costs, benefits, conversion dates, and system constraints.

Architectural Design

It is also known as high level design that focuses on the design of system architecture. It describes the structure and behavior of the system. It defines the structure and relationship between various modules of system development process.

Detailed Design

It follows Architectural design and focuses on development of each module.

Conceptual Data Modeling

It is representation of organizational data which includes all the major entities and relationship. System analysts develop a conceptual data model for the current system that supports the scope and requirement for the proposed system.

The main aim of conceptual data modeling is to capture as much meaning of data as possible. Most organization today use conceptual data modeling using E-R model which uses special notation to represent as much meaning about data as possible.

Entity Relationship Model

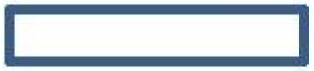
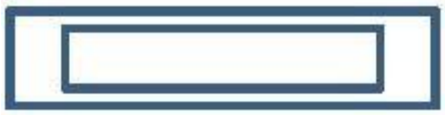
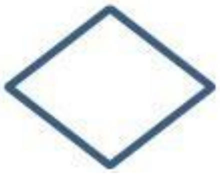
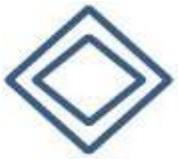
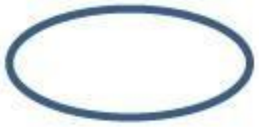
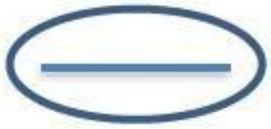
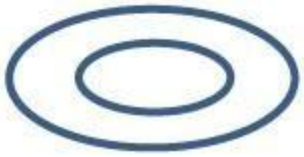
It is a technique used in database design that helps describe the relationship between various entities of an organization.

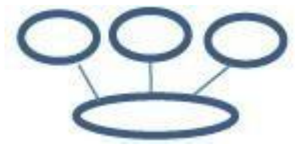
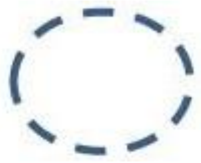
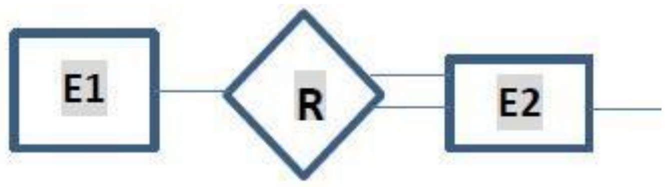
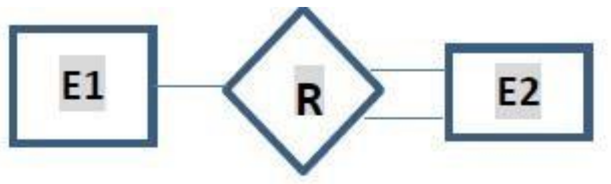
Terms used in E-R model

- **ENTITY** − It specifies distinct real world items in an application. For example: vendor, item, student, course, teachers, etc.
- **RELATIONSHIP** − They are the meaningful dependencies between entities. For example, vendor supplies items, teacher teaches courses, then supplies and course are relationship.
- **ATTRIBUTES** − It specifies the properties of relationships. For example, vendor code, student name. Symbols used in E-R model and their respective meanings −

The following table shows the symbols used in E-R model and their significance −

Symbol	Meaning

	Entity
	Weak Entity
	Relationship
	Identity Relationship
	Attributes
	Key Attributes
	Multivalued

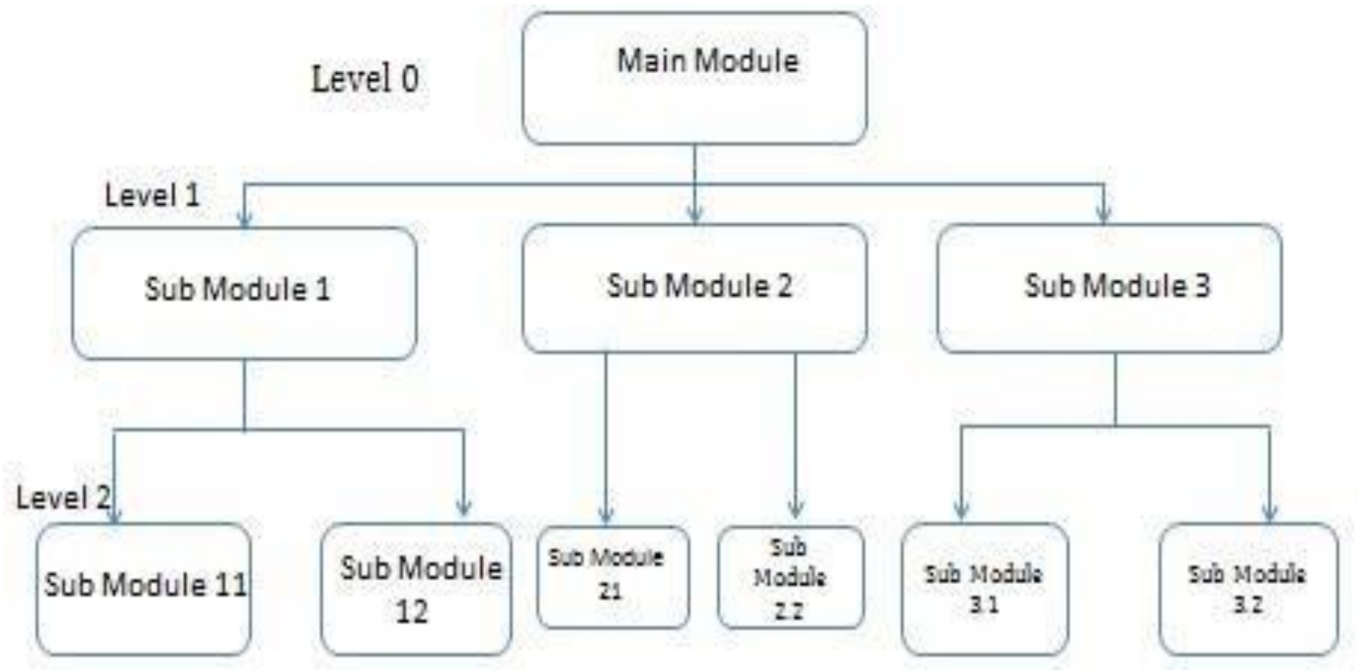
	Composite Attribute
	Derived Attributes
	Total Participation of E2 in R
	Cardinality Ratio 1:N for E1:E2 in R

Three types of relationships can exist between two sets of data: one-to-one, one-to-many, and many-to-many.

Top-Down Strategy

The top-down strategy uses the modular approach to develop the design of a system. It is called so because it starts from the top or the highest-level module and moves towards the lowest level modules.

In this technique, the highest-level module or main module for developing the software is identified. The main module is divided into several smaller and simpler submodules or segments based on the task performed by each module. Then, each submodule is further subdivided into several submodules of next lower level. This process of dividing each module into several submodules continues until the lowest level modules, which cannot be further subdivided, are not identified.



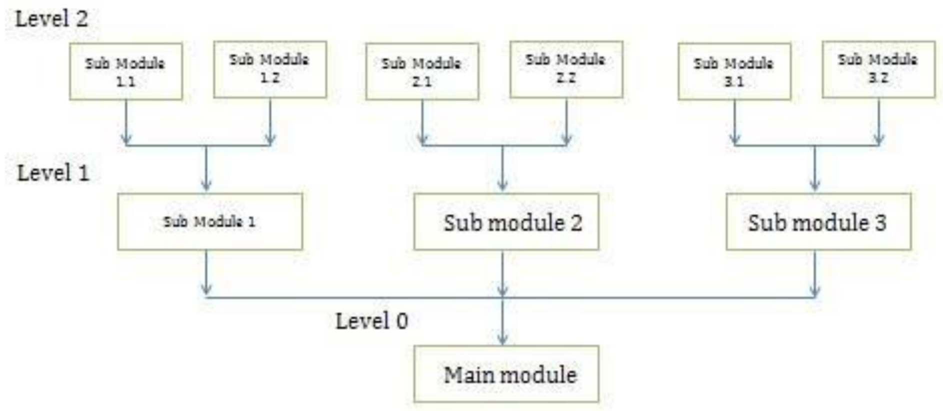
Bottom-Up Strategy

Bottom-Up Strategy follows the modular approach to develop the design of the system. It is called so because it starts from the bottom or the most basic level modules and moves towards the highest level modules.

In this technique,

- The modules at the most basic or the lowest level are identified.

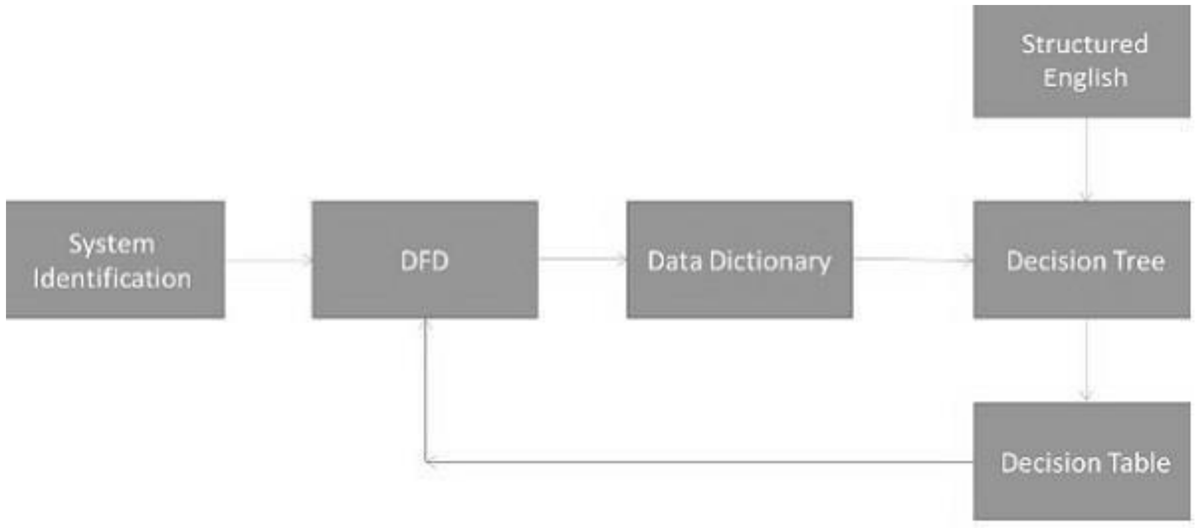
- These modules are then grouped together based on the function performed by each module to form the next higher-level modules.
- Then, these modules are further combined to form the next higher-level modules.
- This process of grouping several simpler modules to form higher level modules continues until the main module of system development process is achieved.



Structured Design

Structured design is a data-flow based methodology that helps in identifying the input and output of the developing system. The main objective of structured design is to minimize the complexity and increase the modularity of a program. Structured design also helps in describing the functional aspects of the system.

In structured designing, the system specifications act as a basis for graphically representing the flow of data and sequence of processes involved in a software development with the help of DFDs. After developing the DFDs for the software system, the next step is to develop the structure chart.



Modularization

Structured design partitions the program into small and independent modules. These are organized in top down manner with the details shown in bottom.

Thus, structured design uses an approach called Modularization or decomposition to minimize the complexity and to manage the problem by subdividing it into smaller segments.

Advantages

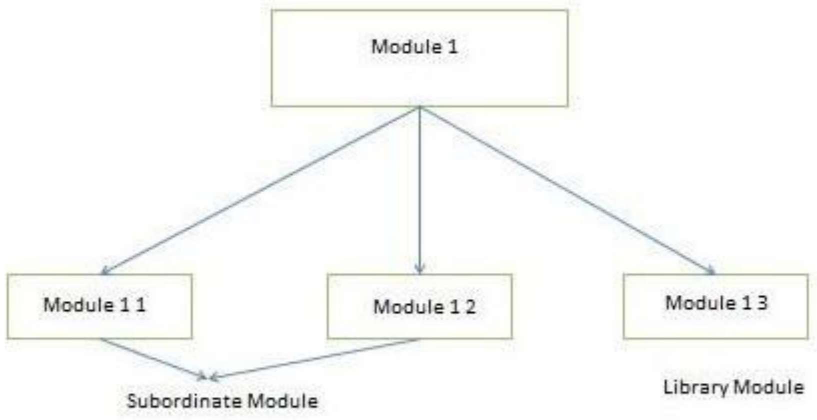
- Critical interfaces are tested first.
- It provide abstraction.
- It allows multiple programmers to work simultaneously.
- It allows code reuse.
- It provides control and improves morale.
- It makes identifying structure easier.

Structured Charts

Structured charts are a recommended tool for designing a modular, top down systems which define the various modules of system development and the relationship between each module. It shows the system module and their relationship between them.

It consists of diagram consisting of rectangular boxes that represent the modules, connecting arrows, or lines.

- **Control Module** − It is a higher-level module that directs lower-level modules, called **subordinate modules**.
- **Library Module** − It is a reusable module and can be invoked from more than one point in the chart.



We have two different approaches to design a structured chart −

- **Transform-Centered Structured Charts** − They are used when all the transactions follow same path.
- **Transaction–Centered Structured Charts** − They are used when all the transactions do not follow the same path.

Objectives of using Structure Flowcharts

- To encourage a top-down design.
- To support the concept of modules and identify the appropriate modules.
- To show the size and complexity of the system.
- To identify the number of readily identifiable functions and modules within each function.
- To depict whether each identifiable function is a manageable entity or should be broken down into smaller components.

Difference between Function Oriented Design and Object Oriented Design

Function-oriented design focuses on defining and organizing functions to perform specific tasks, starting with a high-level description and refining it step-by-step. It uses a top-down approach and often relies on structured analysis and data flow diagrams. On the other hand, object-oriented design emphasizes the data to be manipulated, organizing the software around objects that combine data and behavior. This approach uses a bottom-up strategy, beginning with identifying objects and classes, and often employs UML for design.

What is Function Oriented Design?
Function-oriented design is the result of focusing attention on the function of the program. This is based on the stepwise refinement. Stepwise refinement is based on the iterative procedural decomposition. Stepwise refinement is a top-down strategy where a program is refined as a hierarchy of increasing levels of detail.

We start with a high-level description of what the program does. Then, in each step, we take one part of our high-level description and refine it. Refinement is actually a process of elaboration. The process should proceed from a highly conceptual model to lower-level details. The refinement of each module is done until we reach the statement level of our programming language.

What is Object Oriented Design?
Object-oriented design is the result of focusing attention not on the function performed by the program, but instead on the data that are to be manipulated by the program. Thus, it is orthogonal to function-oriented design. The object-oriented design begins with an examination of real-world "things". These things have individual characteristics in terms of their attributes and behaviors.

Objects are independent entities that may readily be changed because all state and representation information is held within the object itself. Objects may be distributed and may execute sequentially or in parallel. Object-oriented technology contains the following three keywords -

1. **Objects:** Software packages are designed and developed to correspond with real-world entities that contain all the data and services to function as their associated entity messages.
2. **Communication:** Communication mechanisms are established that provide how objects work together.
3. **Methods:** Methods are services that objects perform to satisfy the functional requirements of the problem domain. Objects request services of the other objects through messages.

Difference Between Function Oriented Design and Object Oriented Design

COMPARISON FACTORS	FUNCTION ORIENTED DESIGN	OBJECT ORIENTED DESIGN
Abstraction	The basic abstractions, which are given to the user, are real world functions.	The basic abstractions are not the real world functions but are the data abstraction where the real world entities are represented.
Function	Functions are grouped together by which a higher level function is obtained.	Function are grouped together on the basis of the data they operate since the classes are associated with their methods.
execute	carried out using structured analysis and structured design i.e, data flow diagram	Carried out using UML
State information	In this approach the state information is often represented in a centralized shared memory.	In this approach the state information is not represented in a centralized memory but is implemented or distributed among the objects of the system.
Approach	It is a top down approach.	It is a bottom up approach.

COMPARISON FACTORS	FUNCTION ORIENTED DESIGN	OBJECT ORIENTED DESIGN
Begins basis	Begins by considering the use case diagrams and the scenarios.	Begins by identifying objects and classes.
Decompose	In function oriented design we decompose in function/procedure level.	We decompose in class level.
Use	This approach is mainly used for computation sensitive application.	This approach is mainly used for evolving system which mimics a business or business case.