1. What is modeling?

A model is an abstraction of something for the purpose of understanding it before building it. Because, real systems that we want to study are generally very complex. In order to understand the real system, we have to simplify the system. So a model is an abstraction that hides the non-essential characteristics of a system and highlights those characteristics, which are pertinent to understand it.

Most modeling techniques used for analysis and design involve graphic languages. These graphic languages are made up of sets of symbols. So, the symbols are used according to certain rules of methodology for communicating the complex relationships of information more clearly than descriptive text.

Modeling is used frequently, during many of the phases of the software life cycle such as analysis, design and implementation. Modeling like any other object-oriented development, is an iterative process.

2. Why do we model?

Before constructing anything, a designer first build a model. The main reasons for constructing models include:
• To test a physical entity before actually building it.
• To set the stage for communication between customers and developers.
• For visualization i.e. for finding alternative representations.
• For reduction of complexity in order to understand it.

3. Object Modeling Techniques:

The object modeling techniques is an methodology of object oriented analysis, design and implementation that focuses on creating a model of objects from the real world and then to use this model to develop object–oriented software. objectmodeling technique, OMT was developed by James Rambaugh. Now-a-days, OMT is one of the most popular object oriented development techniques. It is primarily used by system and software developers to support full life cycle development while targeting object oriented implementations.

OMT has proven itself easy to understand, to draw and to use. It is very successful in many application domains: telecommunication, transportation, compilers etc. The popular object modelingtechnique are used in many real world problems. The object-oriented paradigm using the OMT spans the entire development cycle, so there is no need to transform one type of model to another.

Phases of OMT

The OMT methodology covers the full software development life cycle. The methodology has the following phase.

1. **Analysis -** Analysis is the first phase of OMT methodology. The aim of analysis phase is to build a model of the real world situation to show its important properties and domain. This phase is concerned with preparation of precise and correct modelling of the real world. The analysis phase starts with defining a problem statement which includes a set of goals. This problem statement is then expanded into three models; an object model, a dynamic model and a functional model. The object model shows the static data structure or skeleton of the real world system and divides the whole application into objects. In others words, this model represents the artifacts of the system. The dynamic model represents the interaction between artifacts above designed represented as events, states and transitions. The functional model represents the methods of the system from the data flow perspective. The analysis phase generates object model diagrams, state diagrams, event flow diagrams and data flow diagrams.

2. **System design** - The system design phase comes after the analysis phase. System design phase determines the overall system architecture using subsystems, concurrent tasks and data storage. During system design, the high level structure of the system is designed. The decisions made during system design are:
    - The system is organized in to sub-systems which are then allocated to processes and tasks, taking into account concurrency and collaboration.
    - Persistent data storage is established along with a strategy to manage shared or global information.
    - Boundary situations are checked to help guide trade off priorities.

3. **Object design** - The object design phase comes after the system design phase is over. Here the implementation plan is developed. Object design is concerned with fully classifying the existing and remaining classes, associations, attributes and operations necessary for implementing a solution to the problem. In object design:
    - Operations and data structures are fully defined along with any internal objects needed for implementation.
    - Class level associations are determined.
    - Issues of inheritance, aggregation, association and default values are checked.

4. **Implementation -** Implementation pahse of the OMT is a matter of translating the design in to a programming language constructs. It is important to have good software engineering practice so that the design phase is smoothly translated in to the implementation phase. Thus while selecting programming language all constructs should be kept in mind for following noteworthy points.

- To increase flexibility.
- To make amendments easily.
- For the design traceability.
- To increase efficiency.

OMT Methodology use three kinds of model to describe system:

- Object Model
- Dynamic Model
- Functional Model

1. **Object Model** **:**The object model visualizes the elements in a software application in terms of objects.

Object

An object is a real-world element in an object–oriented environment that may have a physical or a conceptual existence. Each object has −

- Identity that distinguishes it from other objects in the system.
- State that determines the characteristic properties of an object as well as the values of the properties that the object holds.
- Behavior that represents externally visible activities performed by an object in terms of changes in its state.

Objects can be modelled according to the needs of the application. An object may have a physical existence, like a customer, a car, etc.; or an intangible conceptual existence, like a project, a process, etc.

Class

A class represents a collection of objects having same characteristic properties that exhibit common behaviour. It gives the blueprint or description of the objects that can be created from it. Creation of an object as a member of a class is called instantiation. Thus, object is an instance of a class.

The constituents of a class are −

- A set of attributes for the objects that are to be instantiated from the class. Generally, different objects of a class have some difference in the values of the attributes. Attributes are often referred as class data.
- A set of operations that portray the behaviour of the objects of the class. Operations are also referred as functions or methods.

Example

Let us consider a simple class, Circle, that represents the geometrical figure circle in a two–dimensional space. The attributes of this class can be identified as follows −

- x–coord, to denote x–coordinate of the center
- y–coord, to denote y–coordinate of the center
- a, to denote the radius of the circle

Some of its operations can be defined as follows −

- findArea(), method to calculate area
- findCircumference(), method to calculate circumference
- scale(), method to increase or decrease the radius

During instantiation, values are assigned for at least some of the attributes. If we create an object my_circle, we can assign values like x-coord : 2, y-coord : 3, and a : 4 to depict its state. Now, if the operation scale() is performed on my_circle with a scaling factor of 2, the value of the variable a will become 8. This operation brings a change in the state of my_circle, i.e., the object has exhibited certain behavior.

The benefits of using the object model are −

- It helps in faster development of software.
- It is easy to maintain. Suppose a module develops an error, then a programmer can fix that particular module, while the other parts of the software are still up and running.
- It supports relatively hassle-free upgrades.
- It enables reuse of objects, designs, and functions.
- It reduces development risks, particularly in integration of complex systems.

2. Dynamic Model:

- Dynamic model describe those aspect of a system concerned with time and the sequencing of operations, events that mark changes, sequences of events, State that define the context for events and the organization of events and states. The dynamic model capture control that aspects of a system that describe the sequences of operations that occur, without regard for what the operation do, what they operate on or how they are implemented.
- Represented by state diagrams
- Dynamic model shows the time dependent behaviour of the system and the objects in it.
- Begin analysis looking for events- externally visible stimuli and responses.
- We need to perform the following steps while constructing a dynamic model:

    a. Prepare scenario of typical interaction sequences
    b. Identify events between objects
    c. Prepare an event trace for each scenario
    d. Build a state diagram
    e. Match events between objects to verify consistency

3. <u>Functional Model:</u>

- The functional model describe those aspect of a system that are concerned with transformation of values: functions, mapping, constraints and functional dependencies. The functional model captures what a system does, without regard how or when it is done.
- Functional model shows how values are computed without regard for sequencing, decision or object structure.
- This model shows which value depend on which other values and the functions that relate them.
- The following steps are performed in constructing a functional model:

  a. Identify input and output values
  b. Build data flow diagram showing functional dependencies,
  c. Describe function
  d. Identify constraints,
  e. Specify optimization criterion

Data Flow Diagrams

Functional Modelling is represented through a hierarchy of DFDs. The DFD is a graphical representation of a system that shows the inputs to the system, the processing upon the inputs, the outputs of the system as well as the internal data stores. DFDs illustrate the series of transformations or computations performed on the objects or the system, and the external controls and objects that affect the transformation.

Rumbaugh et al. have defined DFD as, “A data flow diagram is a graph which shows the flow of data values from their sources in objects through processes that transform them to their destinations on other objects.”

The four main parts of a DFD are −

- Processes,
- Data Flows,
- Actors, and
- Data Stores.

The other parts of a DFD are −

- Constraints, and
- Control Flows.

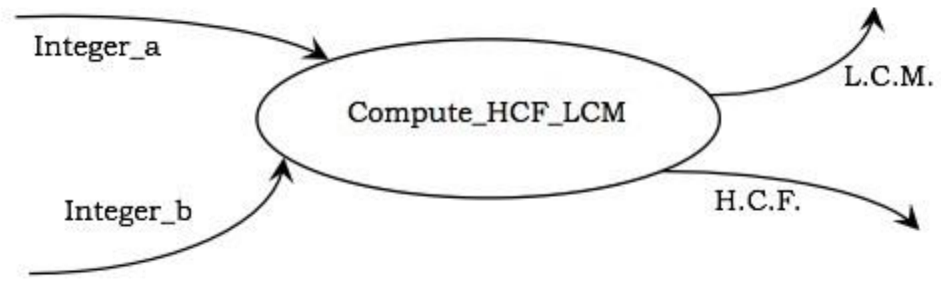
Features of a DFD

Processes

Processes are the computational activities that transform data values. A whole system can be visualized as a high-level process. A process may be further divided into smaller components. The lowest-level process may be a simple function.

Representation in DFD − A process is represented as an ellipse with its name written inside it and contains a fixed number of input and output data values.

Example − The following figure shows a process Compute_HCF_LCM that accepts two integers as inputs and outputs their HCF (highest common factor) and LCM (least common multiple).



Data Flows

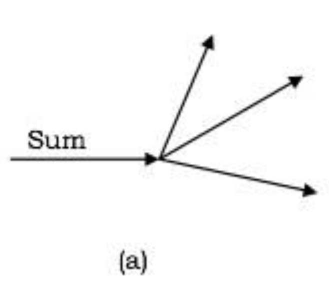
Data flow represents the flow of data between two processes. It could be between an actor and a process, or between a data store and a process. A data flow denotes the value of a data item at some point of the computation. This value is not changed by the data flow.

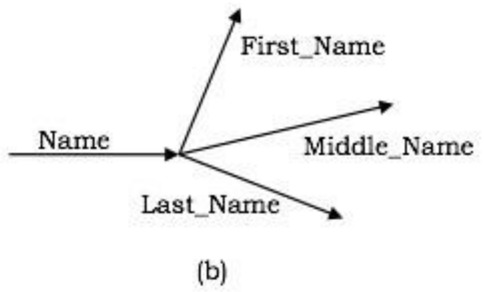
Representation in DFD − A data flow is represented by a directed arc or an arrow, labelled with the name of the data item that it carries.

In the above figure, Integer_a and Integer_b represent the input data flows to the process, while L.C.M. and H.C.F. are the output data flows.

A data flow may be forked in the following cases −

- The output value is sent to several places as shown in the following figure. Here, the output arrows are unlabelled as they denote the same value.
- The data flow contains an aggregate value, and each of the components is sent to different places as shown in the following figure. Here, each of the forked components is labelled.



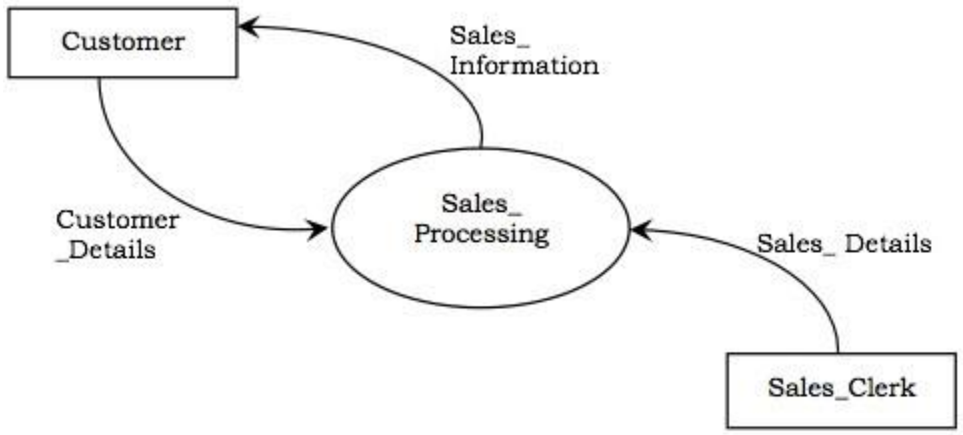


Actors

Actors are the active objects that interact with the system by either producing data and inputting them to the system, or consuming data produced by the system. In other words, actors serve as the sources and the sinks of data.

Representation in DFD − An actor is represented by a rectangle. Actors are connected to the inputs and outputs and lie on the boundary of the DFD.

Example − The following figure shows the actors, namely, Customer and Sales_Clerk in a counter sales system.

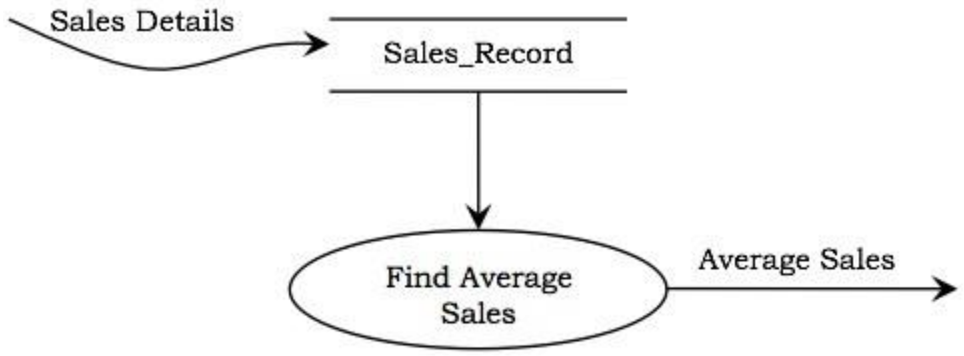


Data Stores

Data stores are the passive objects that act as a repository of data. Unlike actors, they cannot perform any operations. They are used to store data and retrieve the stored data. They represent a data structure, a disk file, or a table in a database.

Representation in DFD − A data store is represented by two parallel lines containing the name of the data store. Each data store is connected to at least one process. Input arrows contain information to modify the contents of the data store, while output arrows contain information retrieved from the data store. When a part of the information is to be retrieved, the output arrow is labelled. An unlabelled arrow denotes full data retrieval. A two-way arrow implies both retrieval and update.

Example − The following figure shows a data store, Sales_Record, that stores the details of all sales. Input to the data store comprises of details of sales such as item, billing amount, date, etc. To find the average sales, the process retrieves the sales records and computes the average.



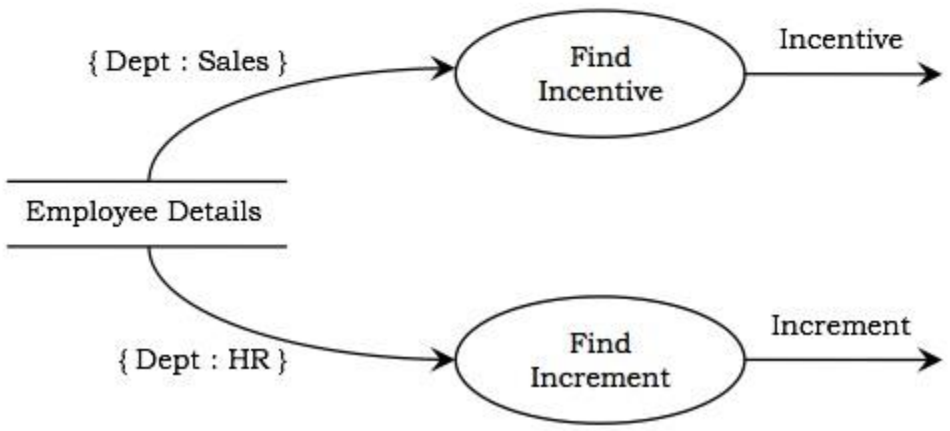
Constraints

Constraints specify the conditions or restrictions that need to be satisfied over time. They allow adding new rules or modifying existing ones. Constraints can appear in all the three models of object-oriented analysis.

- In Object Modelling, the constraints define the relationship between objects. They may also define the relationship between the different values that an object may take at different times.
- In Dynamic Modelling, the constraints define the relationship between the states and events of different objects.
- In Functional Modelling, the constraints define the restrictions on the transformations and computations.

Representation − A constraint is rendered as a string within braces.

Example − The following figure shows a portion of DFD for computing the salary of employees of a company that has decided to give incentives to all employees of the sales department and increment the salary of all employees of the HR department. It can be seen that the constraint {Dept:Sales} causes incentive to be calculated only if the department is sales and the constraint {Dept:HR} causes increment to be computed only if the department is HR.

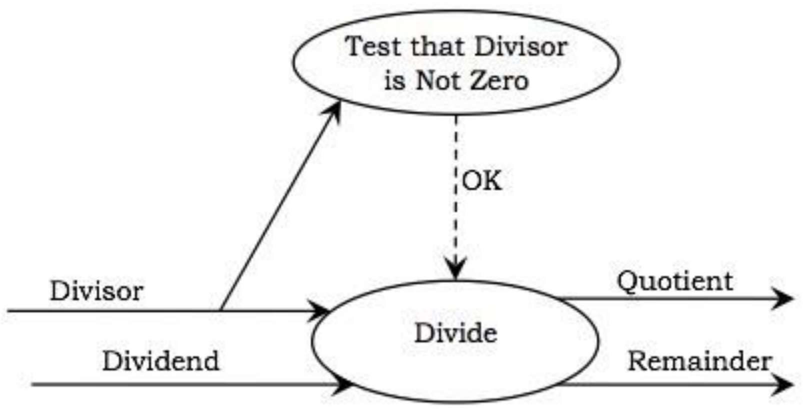


Control Flows

A process may be associated with a certain Boolean value and is evaluated only if the value is true, though it is not a direct input to the process. These Boolean values are called the control flows.

Representation in DFD − Control flows are represented by a dotted arc from the process producing the Boolean value to the process controlled by them.

Example − The following figure represents a DFD for arithmetic division. The Divisor is tested for non-zero. If it is not zero, the control flow OK has a value True and subsequently the Divide process computes the Quotient and the Remainder.



Developing the DFD Model of a System

In order to develop the DFD model of a system, a hierarchy of DFDs are constructed. The top-level DFD comprises of a single process and the actors interacting with it.

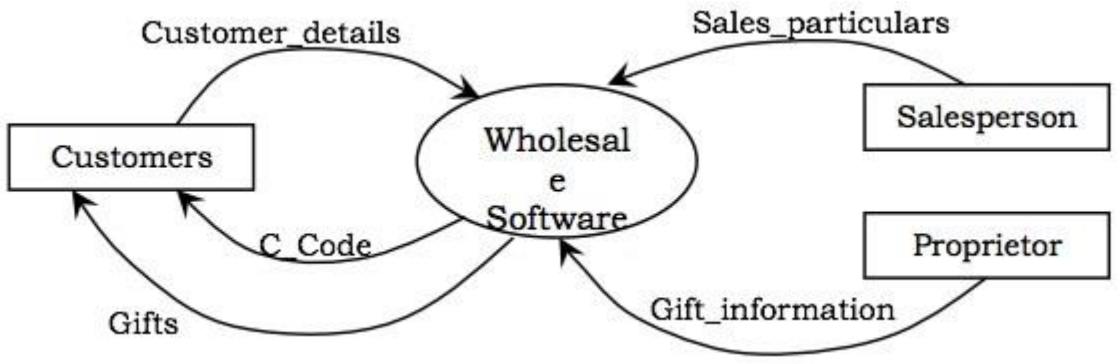
At each successive lower level, further details are gradually included. A process is decomposed into sub-processes, the data flows among the sub-processes are identified, the control flows are determined, and the data stores are defined. While decomposing a process, the data flow into or out of the process should match the data flow at the next level of DFD.

Example − Let us consider a software system, Wholesaler Software, that automates the transactions of a wholesale shop. The shop sells in bulks and has a clientele comprising of merchants and retail shop owners. Each customer is asked to register with his/her particulars and is given a unique customer code, C_Code. Once a sale is done, the shop registers its details and sends the goods for dispatch. Each year, the shop distributes Christmas gifts to its customers, which comprise of a silver coin or a gold coin depending upon the total sales and the decision of the proprietor.

The functional model for the Wholesale Software is given below. The figure below shows the top-level DFD. It shows the software as a single process and the actors that interact with it.

The actors in the system are −

- Customers
- Salesperson
- Proprietor



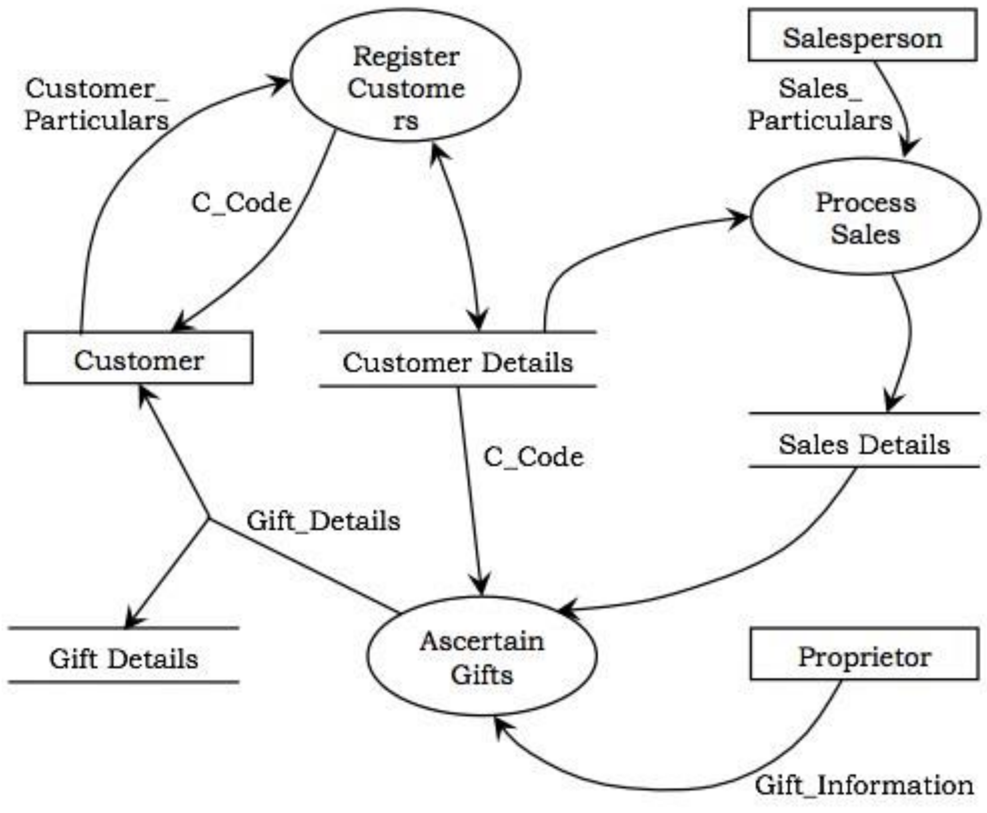
In the next level DFD, as shown in the following figure, the major processes of the system are identified, the data stores are defined and the interaction of the processes with the actors, and the data stores are established.

In the system, three processes can be identified, which are −

- Register Customers
- Process Sales
- Ascertain Gifts

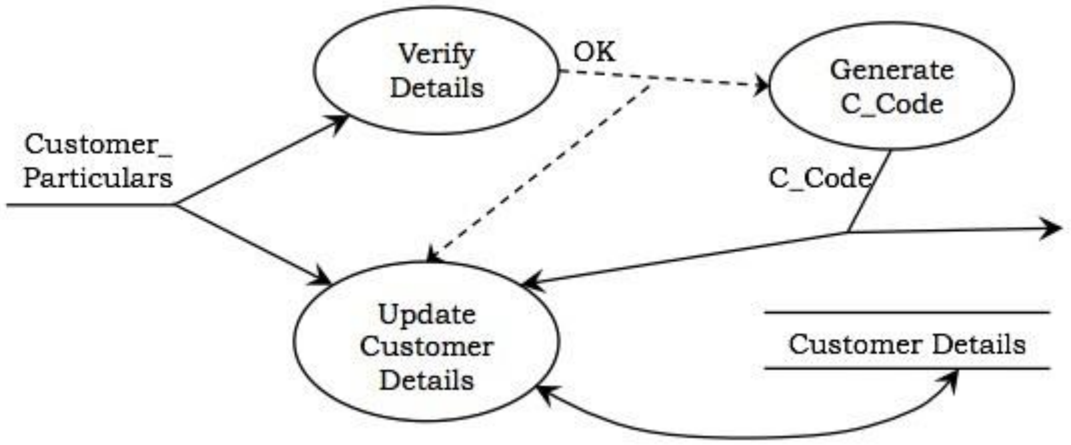
The data stores that will be required are −

- Customer Details
- Sales Details
- Gift Details



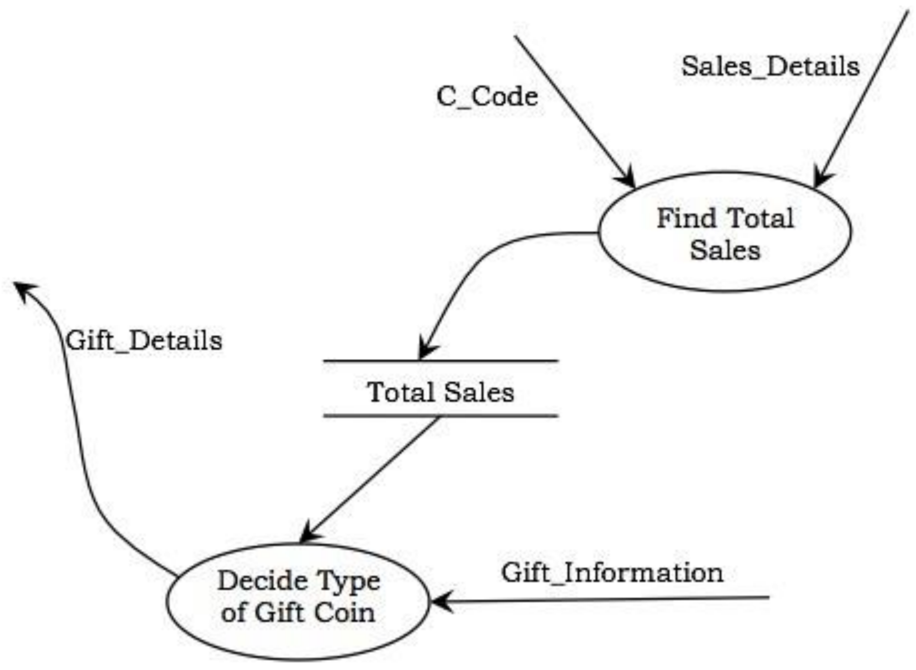
DFD of Wholesale Software

The following figure shows the details of the process Register Customer. There are three processes in it, Verify Details, Generate C_Code, and Update Customer Details. When the details of the customer are entered, they are verified. If the data is correct, C_Code is generated and the data store Customer Details is updated.



DFD of Customer Process

The following figure shows the expansion of the process Ascertain Gifts. It has two processes in it, Find Total Sales and Decide Type of Gift Coin. The Find Total Sales process computes the yearly total sales corresponding to each customer and records the data. Taking this record and the decision of the proprietor as inputs, the gift coins are allotted through Decide Type of Gift Coin process.



DFD of Gift Process

Advantages and Disadvantages of DFD

Advantages	**Disadvantages**
DFDs depict the boundaries of a system and hence are helpful in portraying the relationship between the external objects and the processes within the system.	DFDs take a long time to create, which may not be feasible for practical purposes.
They help the users to have a knowledge about the system.	DFDs do not provide any information about the time-dependent behavior, i.e., they do not specify when the transformations are done.
The graphical representation serves as a blueprint for the programmers to develop a system.	They do not throw any light on the frequency of computations or the reasons for computations.
DFDs provide detailed information about the system processes.	The preparation of DFDs is a complex process that needs considerable expertise. Also, it is

	difficult for a non-technical person to understand.
They are used as a part of the system documentation.	The method of preparation is subjective and leaves ample scope to be imprecise.

Relationship between Object, Dynamic, and Functional Models

The Object Model, the Dynamic Model, and the Functional Model are complementary to each other for a complete Object-Oriented Analysis.

- Object modelling develops the static structure of the software system in terms of objects. Thus it shows the “doers” of a system.
- Dynamic Modelling develops the temporal behavior of the objects in response to external events. It shows the sequences of operations performed on the objects.
- Functional model gives an overview of what the system should do.

Functional Model and Object Model

The four main parts of a Functional Model in terms of object model are −

- **Process** − Processes imply the methods of the objects that need to be implemented.
- **Actors** − Actors are the objects in the object model.
- **Data Stores** − These are either objects in the object model or attributes of objects.
- **Data Flows** − Data flows to or from actors represent operations on or by objects. Data flows to or from data stores represent queries or updates.

Functional Model and Dynamic Model

The dynamic model states when the operations are performed, while the functional model states how they are performed and which arguments are needed. As actors are active objects, the dynamic model has to specify when it acts. The data stores are passive objects and they only respond to updates and queries; therefore the dynamic model need not specify when they act.

Object Model and Dynamic Model

The dynamic model shows the status of the objects and the operations performed on the occurrences of events and the subsequent changes in states. The state of the object as a result of the changes is shown in the object model.

Structure Analysis and Structure Design (SA/SD)

In software engineering, **structured analysis** (SA) and **structured design** (SD) are methods for analyzing business requirements and developing specifications for converting practices into computer programs, hardware configurations, and related manual procedures.

Structured analysis and design techniques are fundamental tools of systems analysis. Structured analysis consists of interpreting the system concept (or real world situations) into data and control terminology represented by data flow diagrams. The flow of data and control from bubble to the data store to bubble can be difficult to track and the number of bubbles can increase.

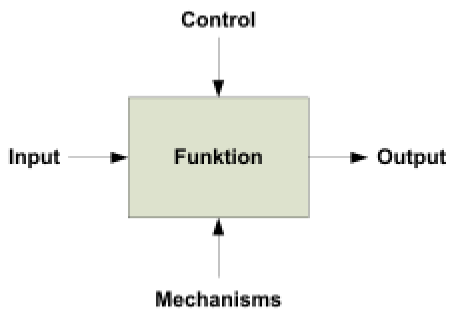


Fig 1: SA/SD basic element

Structured Analysis Tools

Structured Analysis views a system from the perspective of the data flowing through it. The function of the system is described by processes that transform the data flows. Structured analysis takes advantage of information hiding through successive decomposition (or top down) analysis. This allows attention to be focused on pertinent details and avoids confusion from looking at irrelevant details. As the level of detail increases, the breadth of information is reduced. The result of structured analysis is a set of related graphical diagrams, process descriptions, and data definitions. They describe the transformations that need to take place and the data required to meet a system's functional requirements**.**

various tools and techniques are used for system development. They are −

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

1. Data Flow Diagrams (DFD) or Bubble Chart

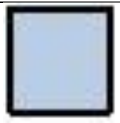
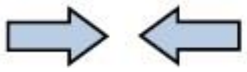
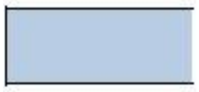
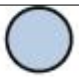
It is a technique developed by Larry Constantine to express the requirements of system in a graphical form.

- 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
Open Rectangle		Process transforming data flow
circle		Data store

types of DFD

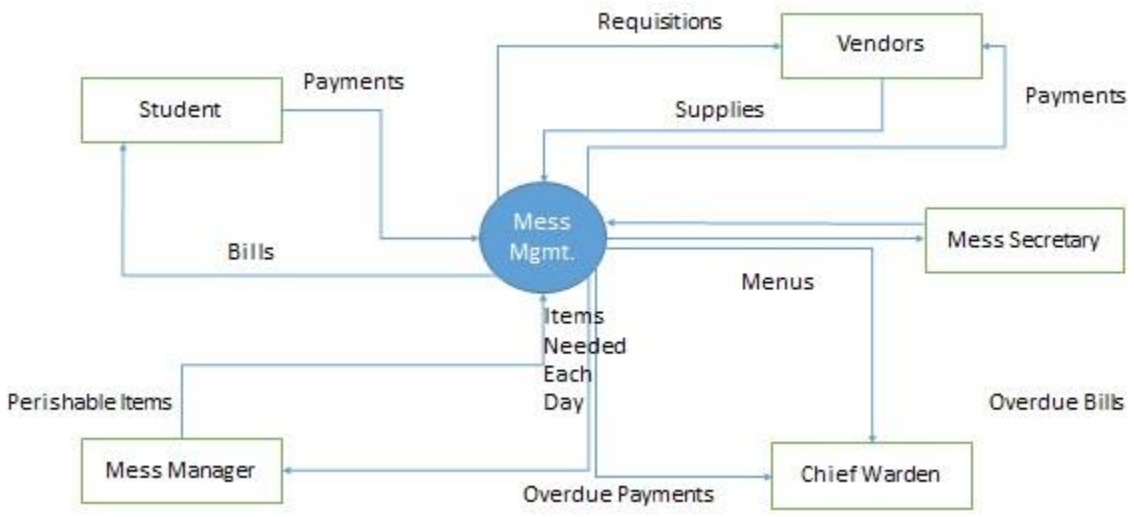
DFDs are of two types: Physical DFD and Logical DFD. The following table lists the points that differentiate a physical DFD from a logical DFD.

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.

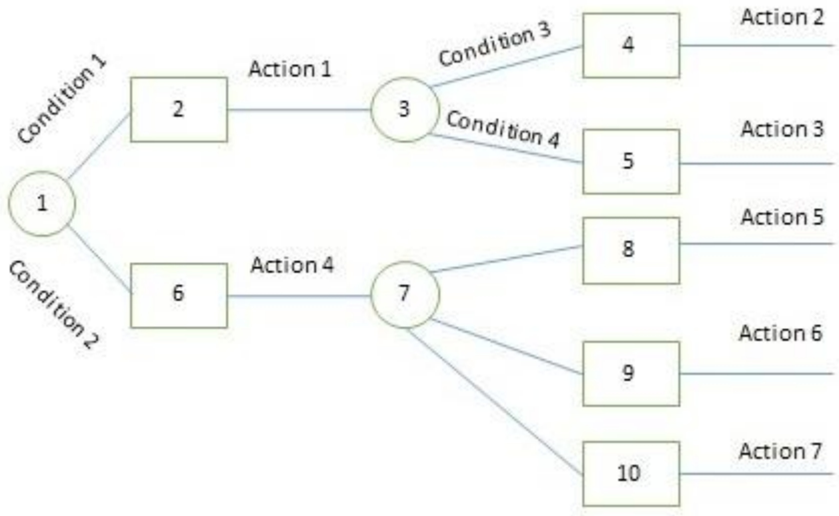
A data dictionary improves the communication between the analyst and the user. It plays an important role in building a database. Most DBMSs have a data dictionary as a standard feature. For example, refer the following table −

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

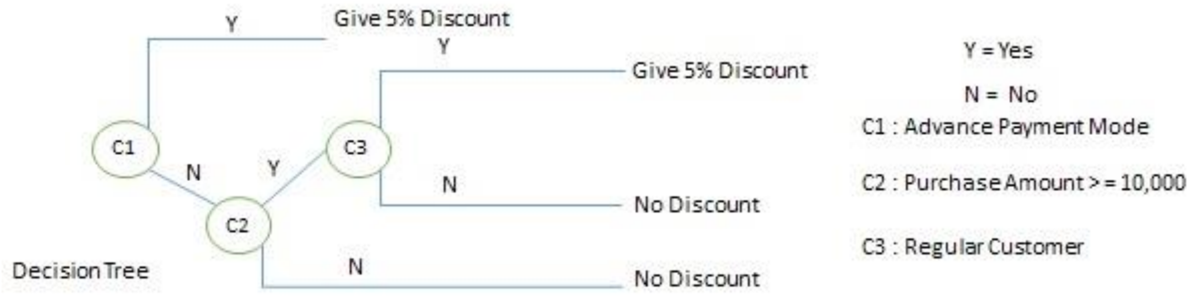
Decision trees are a method for defining complex relationships by describing decisions and avoiding the problems in communication. A decision tree is a diagram that shows alternative actions and conditions within horizontal tree framework. Thus, it depicts which conditions to consider first, second,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



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.

Structured Analysis vs. Object Oriented Analysis

The Structured Analysis/Structured Design (SASD) approach is the traditional approach of software development based upon the waterfall model. The phases of development of a system using SASD are −

- Feasibility Study
- Requirement Analysis and Specification
- System Design
- Implementation
- Post-implementation Review

Now, we will look at the relative advantages and disadvantages of structured analysis approach and object-oriented analysis approach.

Advantages/Disadvantages of Object Oriented Analysis

Advantages	**Disadvantages**
Focuses on data rather than the procedures as in Structured Analysis.	Functionality is restricted within objects. This may pose a problem for systems which are intrinsically procedural or computational in nature.
The principles of encapsulation and data hiding help the developer to develop systems that cannot be tampered by other parts of the system.	It cannot identify which objects would generate an optimal system design.
The principles of encapsulation and data hiding help the developer to develop systems that cannot be tampered by other parts of the system.	The object-oriented models do not easily show the communications between the objects in the system.
It allows effective management of software complexity by the virtue of modularity.	All the interfaces between the objects cannot be represented in a single diagram.
It can be upgraded from small to large systems at a greater ease than in systems following structured analysis.	

Advantages/Disadvantages of Structured Analysis

Advantages	**Disadvantages**
As it follows a top-down approach in contrast to bottom-	In traditional structured analysis models, one

up approach of object-oriented analysis, it can be more easily comprehended than OOA.	phase should be completed before the next phase. This poses a problem in design, particularly if errors crop up or requirements change.
It is based upon functionality. The overall purpose is identified and then functional decomposition is done for developing the software. The emphasis not only gives a better understanding of the system but also generates more complete systems.	The initial cost of constructing the system is high, since the whole system needs to be designed at once leaving very little option to add functionality later.
The specifications in it are written in simple English language, and hence can be more easily analyzed by non-technical personnel.	It does not support reusability of code. So, the time and cost of development is inherently high.

Jackson System Development (JSD)

jackson system development (**JSD**) is a linear software development methodology developed by Michael A. Jackson and John Cameron in the 1980s.

Jackson System Development (JSD) is a method of system development that covers the software life cycle either directly or, by providing a framework into which more specialized techniques can fit. Jackson System Development can start from the stage in a project when there is only a general statement of requirements. However, many projects that have used Jackson System Development actually started slightly later in the life cycle, doing the first steps largely from existing documents rather than directly with the users. The later steps of JSD produce the code of the final system. Jackson's first method, Jackson Structured Programming (JSP), is used to produce the final code. The output of the earlier steps of JSD are a set of program design problems, the design of which is the subject matter of JSP. Maintenance is also addressed by reworking whichever of the earlier steps are appropriate.

Principles of operation

Three basic principles of operation of JSD is that:

- Development must start with describing and modelling the real world, rather than specifying or structuring the function performed by the system. A system made using JSD method performs the simulation of the real world before any direct attention is paid to function or purpose of the system.
- An adequate model of a time-ordered world must itself be time-ordered. Main aim is to map progress in the real world on progress in the system that models it.

- The way of implementing the system is based on transformation of specification into efficient set of processes. These processes should be designed in such a manner that it would be possible to run them on available software and hardware.

JSD steps

When it was originally presented by Jackson in 1982 the method consisted of six steps:

1. Entity/action step
2. Initial model step
3. Interactive function step
4. Information function step
5. System timing step
6. System implementation step

Later, some steps were combined to create a method with only three steps

1. Modelling stage (analysis): with the *entity/action step* and *entity structures step*.
2. Network stage (design): with the *initial model step*, *function step,* and *system timing step*.
3. Implementation stage (realisation): the implementation step.

Modeling stage

In the modeling stage the designer creates a collection of *entity structure diagrams* and identifies the entities in the system, the actions they perform, the time-ordering of the actions in the life of the entities, and the attributes of the actions and entities. Entity structure diagrams use the diagramming notation of Jackson Structured Programming *structure diagrams*. Purpose of these diagrams is to create a full description of the aspects of the system and the organisation. Developers have to decide which things are important and which are not. Good communication between developers and users of the new system is very important.

This stage is the combination of the former entity/action step and the entity structures step.

Network stage

In the network stage a model of the system as a whole is developed and represented as a *system specification diagram* (SSD) (also known as a *network diagram*). Network diagrams show processes (rectangles) and how they communicate with each other, either via *state vector* connections (diamonds) or via *datastream* connections (circles). In this stage is the functionality of the system defined. Each entity becomes a process or program in the network diagram. External programs are later added to the network diagrams. The purpose of these programs is to process input, calculate output and to keep the entity processes up-to-date. The whole system is described with these network diagrams and are completed with descriptions about the data and connections between the processes and programs.

The initial model step specifies a simulation of the real world. The function step adds to this simulation the further executable operations and processes needed to produce output of the system. System timing step provides synchronisation among processes, introduces constraints. This stage is the combination of the former ‘Initial model’ step, the ‘function’ step and the ‘system timing’ step.

Implementation stage

In the implementation stage the abstract network model of the solution is converted into a physical system, represented as a *system implementation diagram* (SID). The SID shows the system as a *scheduler* process that calls modules that implement the processes. Datastreams are represented as calls to inverted processes. Database symbols represent collections of entity state-vectors, and there are special symbols for file buffers (which must be implemented when processes are scheduled to run at different time intervals).

The central concern of implementation step is optimization of the system. It is necessary to reduce the number of processes because it is impossible to provide each process that is contained in specification with its own virtual processor. By means of transformation, processes are combined in order to limit their number to the number of processors.