BACHELOR OF TECHNOLOGY YEAR 3

OBJECT ORIENTED ANALYSIS AND DESIGN

WEEK 4 & 5

FUNCTIONAL MODELLING

Functional Modeling gives the process perspective of the object-oriented analysis model and an overview of what the system is supposed to do. It defines:-

– The *function of the internal processes* in the system with the aid of *Data Flow Diagrams* (**DFDs**)

– Functional derivation of the data values without indicating how they are derived when they are computed, or why they need to be computed.

DATA FLOW DIAGRAMS

Functional Modeling is represented through a hierarchy of DFDs. A **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.

Definition - "*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

- Data Stores

The other parts of a DFD are:-

- Constraints, and
- Control Flows.

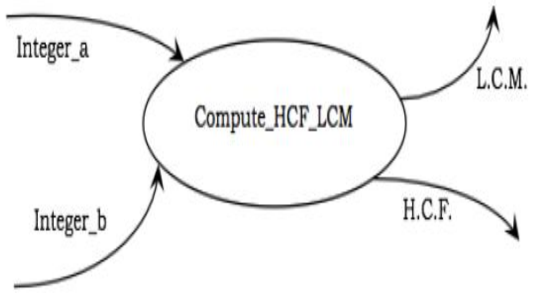
FEATURES OF A DFD

a) 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 of computing LCM *Compute_HCF_LCM* that accepts two integers as inputs i.e. *integer_a* and *integer_b* and outputs their HCF (highest common factor) and LCM (least common multiple).



b) *Data Flows*

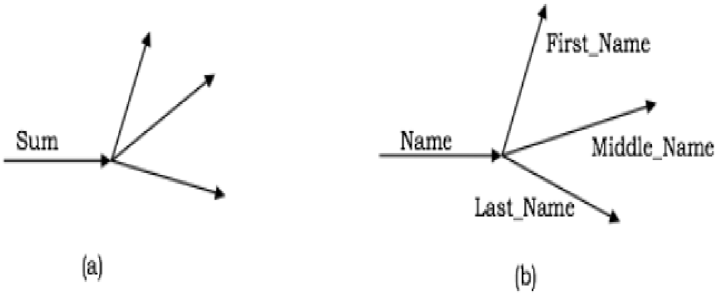
Data flow represents the *flow of data* between two processes. It could be between an actor and a process, or between a data store and a process. A data flow denotes the value of a data item at some point of the computation. This value is not changed by the data flow.

Representation in a DFD: Data flow is represented by a directed arc or an arrow, labeled 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 labeled.

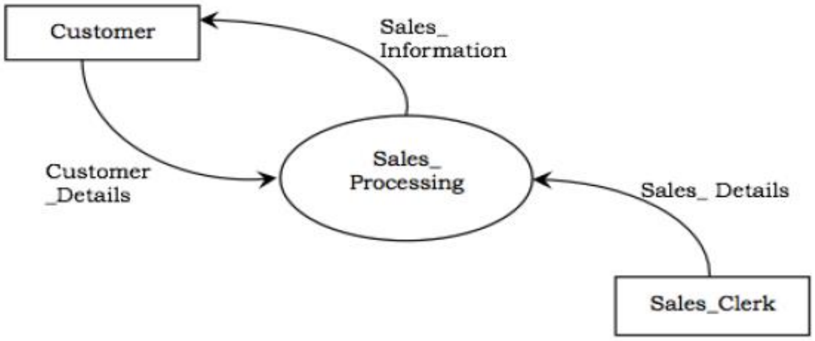


c) Actors

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 sales system.



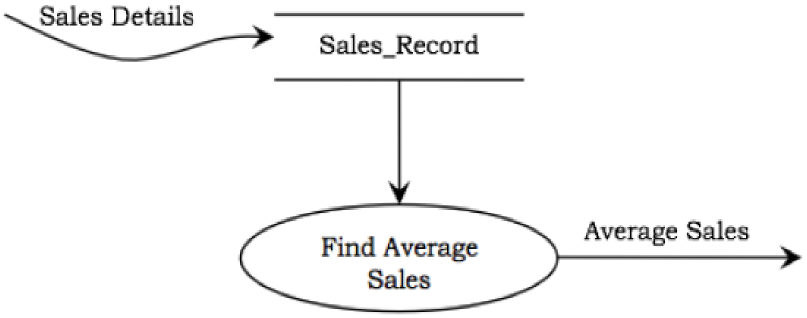
d) Data Stores

Data stores are the passive objects that act as a repository of data. Unlike actors, they cannot perform any operations. They are used for data storage and retrieval of 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 labeled. An unlabelled arrow denotes full data retrieval. A two-way arrow implies retrieval (read) and update (read).

Example: The figure below 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.



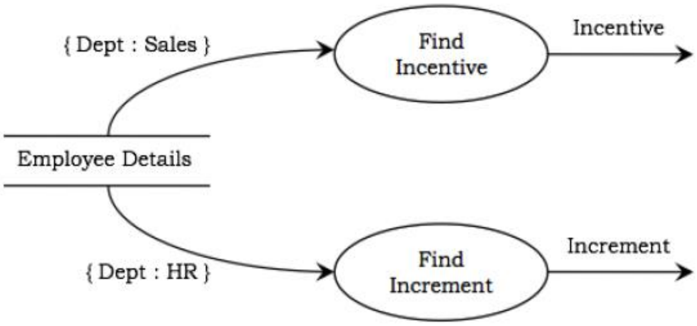
e) Constraints

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 Modeling, 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 Modeling, the constraints define the relationship between the states and events of different objects.
- In Functional Modeling, the constraints define the restrictions on the transformations and computations.

Representation in a DFD: A constraint is rendered as a string within braces.

Example: The following figure shows a portion of DFD for a payroll system 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.

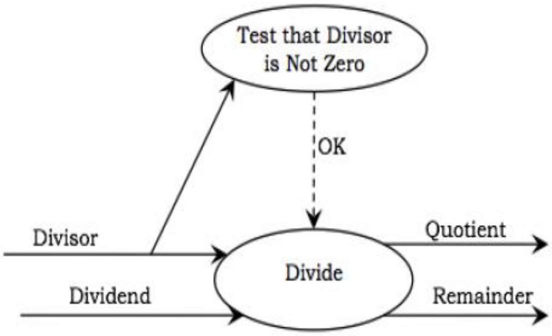


f) Control Flows

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 A DFD MODEL OF A SYSTEM

In order to develop a DFD model of a system, a hierarchy of DFDs is 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.

Note:

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 Wholesaler Software System that automates the transactions of a wholesale shop. The shop sells in bulk 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.

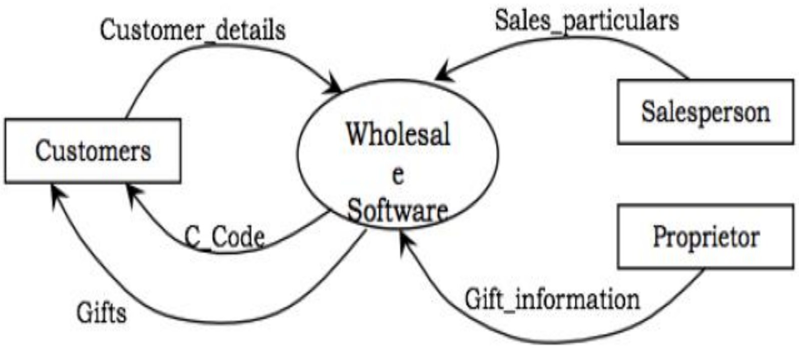
Each year, the shop distributes Christmas gifts to its customers, which comprise of a silver coin or a gold coin depending upon the total sales and the decision of the proprietor.

The functional model for the Wholesale Software below shows the top-level DFD illustrating the software as a single process and the actors that interact with it. This is also called a *Context Diagram*

The actors in the system are:-

- Customers
- Salesperson

- Proprietor



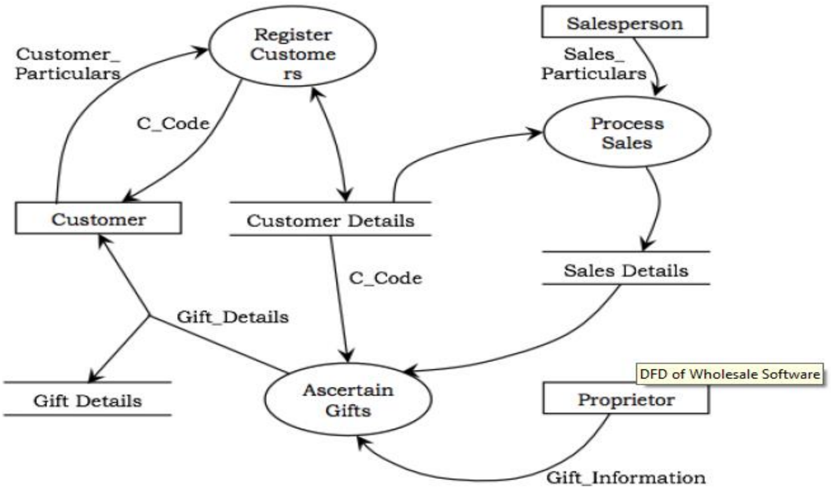
In the next level DFD, as shown in the following figure below, the major processes of the system are identified, the data stores are defined and the interaction of the processes with the actors and the data stores are established.

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

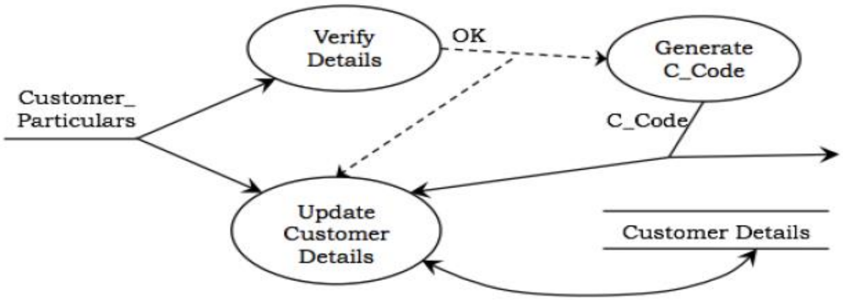


DFD of Wholesale Software

DECOMPOSITION OF THE PROCESSES ILLUSTRATION

a) Decomposition of the Register Customer Process

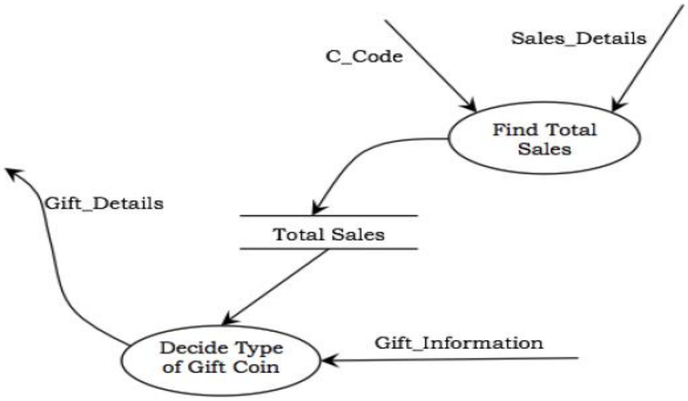
The following figure shows the details of the *process Register Customer*. There are three processes in it, Verify Details, Generate C_Code, and Update Customer Details. When the details of the customer are entered, they are verified. If the data is correct, C_Code is generated and the data store *Customer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



DFD of Customer Process

b) Decomposition of the Ascertain Gifts Process

The following figure shows the decomposition 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 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.
They are used as a part of the system documentation.	Also, it is difficult for a non-technical person to understand.
	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.

a) 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 and contain 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.

b) 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, and the dynamic model has to specify when they act.

The *data stores* are passive objects and they only respond to *updates* and *queries*; therefore the dynamic model need not specify when they act.

c) 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 as shown in the object model

STRUCTURED ANALYSIS VS. OBJECT-ORIENTED ANALYSIS

The Structured Analysis/Structured Design (SASD) approach is the traditional approach of software development is 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 with by other parts of the system.	It cannot identify which objects would generate an optimal system design.
It allows effective management of software complexity by the virtue of modularity.	The object-oriented models do not easily show the communications between the objects in the system.
It can be upgraded from small to large systems at a greater ease than in systems following structured analysis.	All the interfaces between the objects cannot be represented in a single diagram.

Advantages/Disadvantages of Structured Analysis	
Advantages	***Disadvantages***
Since it follows a top-down approach in contrast to bottom-up approach of OOAD, it can be more easily comprehended	In traditional structured analysis models, one phase should be completed before the next phase.

It is based upon functionality. The overall purpose is identified and then functional decomposition is done for developing the software. This 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 are written in simple English language, and hence can be more easily analyzed by non-technical personnel.	It does not support reusability of code thus time and cost of development is inherently high.