Unit 2: Explain Modeling and Software Development Process
Most Essential Learning Competencies: *At the end of the course, you must be able to:*

1. Explain Object Oriented Software Development Process
2. Describe the benefits of the Software Modeling Process

Principles of Object-Oriented Systems

The conceptual framework of object–oriented systems is based upon the object model. There are two categories of elements in an object-oriented system −

Major Elements − By major, it is meant that if a model does not have any one of these elements, it ceases to be object oriented. The four major elements are −

- Abstraction
- Encapsulation
- Modularity
- Hierarchy

Minor Elements − By minor, it is meant that these elements are useful, but not indispensable part of the object model. The three minor elements are −

- Typing
- Concurrency
- Persistence

Abstraction

Abstraction means to focus on the essential features of an element or object in OOP, ignoring its extraneous or accidental properties. The essential features are relative to the context in which the object is being used.

Grady Booch has defined abstraction as follows −

"An abstraction denotes the essential characteristics of an object that distinguish it from all other kinds of objects and thus provide crisply defined conceptual boundaries, relative to the perspective of the viewer."

Example − When a class Student is designed, the attributes enrolment_number, name, course, and address are included while characteristics like pulse_rate and size_of_shoe are eliminated, since they are irrelevant in the perspective of the educational institution.

Encapsulation

Encapsulation is the process of binding both attributes and methods together within a class. Through encapsulation, the internal details of a class can be hidden from outside. The class has methods that provide user interfaces by which the services provided by the class may be used.

Modularity

Modularity is the process of decomposing a problem (program) into a set of modules so as to reduce the overall complexity of the problem. Booch has defined modularity as − "Modularity is the property of a system that has been decomposed into a set of cohesive and loosely coupled modules."

Modularity is intrinsically linked with encapsulation. Modularity can be visualized as a way of mapping encapsulated abstractions into real, physical modules having high cohesion within the modules and their inter–module interaction or coupling is low.

Hierarchy

In Grady Booch's words, "Hierarchy is the ranking or ordering of abstraction". Through hierarchy, a system can be made up of interrelated subsystems, which can have their own subsystems and so on until the smallest level components are reached. It uses the principle of "divide and conquer". Hierarchy allows code reusability.

The two types of hierarchies in OOA are −

- **"IS–A" hierarchy** − It defines the hierarchical relationship in inheritance, whereby from a super-class, a number of subclasses may be derived which may again have subclasses and so on. For example, if we derive a class Rose from a class Flower, we can say that a rose "is–a" flower.
- **"PART–OF" hierarchy** − It defines the hierarchical relationship in aggregation by which a class may be composed of other classes. For example, a flower is composed of sepals, petals, stamens, and carpel. It can be said that a petal is a "part–of" flower.

Typing

According to the theories of abstract data type, a type is a characterization of a set of elements. In OOP, a class is visualized as a type having properties distinct from any other types. Typing is the enforcement of the notion that an object is an instance of a single class or type. It also enforces that objects of different types may not be generally interchanged; and can be interchanged only in a very restricted manner if absolutely required to do so.

The two types of typing are −

- **Strong Typing** − Here, the operation on an object is checked at the time of compilation, as in the programming language Eiffel.
- **Weak Typing** − Here, messages may be sent to any class. The operation is checked only at the time of execution, as in the programming language Smalltalk.

Concurrency

Concurrency in operating systems allows performing multiple tasks or processes simultaneously. When a single process exists in a system, it is said that there is a single thread of control. However, most systems have multiple threads, some active, some waiting for CPU, some suspended, and some terminated. Systems with multiple CPUs inherently permit concurrent threads of control; but systems running on a single CPU use appropriate algorithms to give equitable CPU time to the threads so as to enable concurrency.

In an object-oriented environment, there are active and inactive objects. The active objects have independent threads of control that can execute concurrently with threads of other objects. The active objects synchronize with one another as well as with purely sequential objects.

Persistence

An object occupies a memory space and exists for a particular period of time. In traditional programming, the lifespan of an object was typically the lifespan of the execution of the program that created it. In files or databases, the object lifespan is longer than the duration of the process creating the object. This property by which an object continues to exist even after its creator ceases to exist is known as persistence.

Object Oriented Analysis

We know that the Object-Oriented Modeling (OOM) technique visualizes things in an application by using models organized around objects. Any software development approach goes through the following stages −

- Analysis,
- Design, and
- Implementation.

In object-oriented software engineering, the software developer identifies and organizes the application in terms of object-oriented concepts, prior to their final representation in any specific programming language or software tools.

Phases in Object-Oriented Software Development

The major phases of software development using object–oriented methodology are object-oriented analysis, object-oriented design, and object-oriented implementation.

Object–Oriented Analysis

In this stage, the problem is formulated, user requirements are identified, and then a model is built based upon real–world objects. The analysis produces models on how the desired system should function and how it must be developed. The models do not include any implementation details so that it can be understood and examined by any non–technical application expert.

Object–Oriented Design

Object-oriented design includes two main stages, namely, system design and object design.

System Design

In this stage, the complete architecture of the desired system is designed. The system is conceived as a set of interacting subsystems that in turn is composed of a hierarchy of interacting objects, grouped into classes. System design is done according to both the system analysis model and the proposed system architecture. Here, the emphasis is on the objects comprising the system rather than the processes in the system.

Object Design

In this phase, a design model is developed based on both the models developed in the system analysis phase and the architecture designed in the system design phase. All the classes required are identified. The designer decides whether −

- new classes are to be created from scratch,
- any existing classes can be used in their original form, or
- new classes should be inherited from the existing classes.

The associations between the identified classes are established and the hierarchies of classes are identified. Besides, the developer designs the internal details of the classes and their associations, i.e., the data structure for each attribute and the algorithms for the operations.

Object–Oriented Implementation and Testing

In this stage, the design model developed in the object design is translated into code in an appropriate programming language or software tool. The databases are created and the specific hardware requirements are ascertained. Once the code is in shape, it is tested using specialized techniques to identify and remove the errors in the code

Dynamic Modeling

The dynamic model represents the time–dependent aspects of a system. It is concerned with the temporal changes in the states of the objects in a system. The main concepts are −

- State, which is the situation at a particular condition during the lifetime of an object.
- Transition, a change in the state
- Event, an occurrence that triggers transitions
- Action, an uninterrupted and atomic computation that occurs due to some event, and
- Concurrency of transitions.

A state machine models the behavior of an object as it passes through a number of states in its lifetime due to some events as well as the actions occurring due to the events. A state machine is graphically represented through a state transition diagram.

States and State Transitions

State

The state is an abstraction given by the values of the attributes that the object has at a particular time period. It is a situation occurring for a finite time period in the lifetime of an object, in which it fulfils certain conditions, performs certain activities, or waits for certain events to occur. In state transition diagrams, a state is represented by rounded rectangles.

Parts of a state:

- **Name** − A string differentiates one state from another. A state may not have any name.
- **Entry/Exit Actions** − It denotes the activities performed on entering and on exiting the state.
- **Internal Transitions** − The changes within a state that do not cause a change in the state.
- **Sub–states** − States within states.

Initial and Final States

The default starting state of an object is called its initial state. The final state indicates the completion of execution of the state machine. The initial and the final states are pseudo-states, and may not have the parts of a regular state except name. In state transition diagrams, the initial state is represented by a filled black circle. The final state is represented by a filled black circle encircled within another unfilled black circle.

Transition

A transition denotes a change in the state of an object. If an object is in a certain state when an event occurs, the object may perform certain activities subject to specified conditions and change the state. In this case, a state−transition is said to have occurred. The transition gives the relationship between the first state and the new state. A transition is graphically represented by a solid directed arc from the source state to the destination state.

The five parts of a transition are −

- **Source State** − The state affected by the transition.
- **Event Trigger** − The occurrence due to which an object in the source state undergoes a transition if the guard condition is satisfied.
- **Guard Condition** − A Boolean expression which if True, causes a transition on receiving the event trigger.

- **Action** − An un-interruptible and atomic computation that occurs on the source object due to some event.
- **Target State** − The destination state after completion of transition.

Example

Suppose a person is taking a taxi from place X to place Y. The states of the person may be: Waiting (waiting for taxi), Riding (he has got a taxi and is travelling in it), and Reached (he has reached the destination). The following figure depicts the state transition.

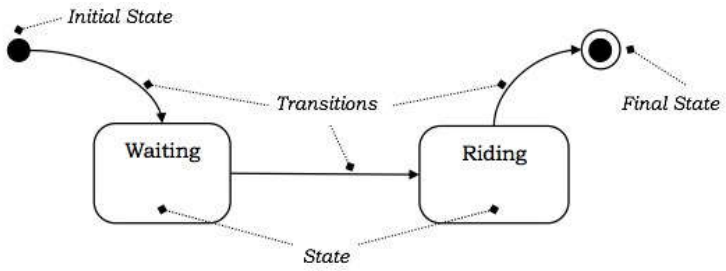


Figure 11-12.1

Events

Events are some occurrences that can trigger state transition of an object or a group of objects. Events have a location in time and space but do not have a time period associated with it. Events are generally associated with some actions.

Examples of events are mouse click, key press, an interrupt, stack overflow, etc.

Events that trigger transitions are written alongside the arc of transition in state diagrams.

Example

Considering the example shown in the above figure, the transition from Waiting state to Riding state takes place when the person gets a taxi. Likewise, the final state is reached, when he reaches the destination. These two occurrences can be termed as events Get_Taxi and Reach_Destination. The following figure shows the events in a state machine.

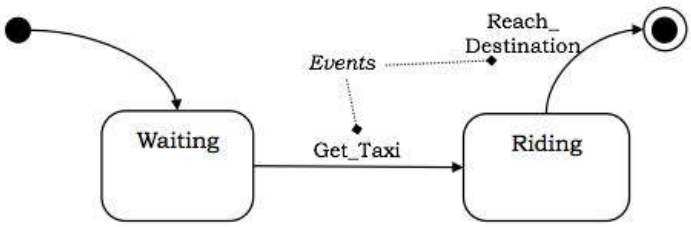


Figure 11-12.2

External and Internal Events

External events are those events that pass from a user of the system to the objects within the system. For example, mouse click or key−press by the user are external events.

Internal events are those that pass from one object to another object within a system. For example, stack overflow, a divide error, etc.

Deferred Events

Deferred events are those which are not immediately handled by the object in the current state but are lined up in a queue so that they can be handled by the object in some other state at a later time.

Event Classes

Event class indicates a group of events with common structure and behavior. As with classes of objects, event classes may also be organized in a hierarchical structure. Event classes may have attributes associated with them, time being an implicit attribute. For example, we can consider the events of departure of a flight of an airline, which we can group into the following class −

Flight_Departs (Flight_No, From_City, To_City, Route)

Actions

Activity

Activity is an operation upon the states of an object that requires some time period. They are the ongoing executions within a system that can be interrupted. Activities are shown in activity diagrams that portray the flow from one activity to another.

Action

An action is an atomic operation that executes as a result of certain events. By atomic, it is meant that actions are un-interruptible, i.e., if an action starts executing, it runs into completion without being interrupted by any event. An action may operate upon an object on which an event has been triggered or on other objects that are visible to this object. A set of actions comprise an activity.

Entry and Exit Actions

Entry action is the action that is executed on entering a state, irrespective of the transition that led into it.

Likewise, the action that is executed while leaving a state, irrespective of the transition that led out of it, is called an exit action.

Scenario

Scenario is a description of a specified sequence of actions. It depicts the behavior of objects undergoing a specific action series. The primary scenarios depict the essential sequences and the secondary scenarios depict the alternative sequences.

Diagrams for Dynamic Modeling

There are two primary diagrams that are used for dynamic modeling −

Interaction Diagrams

Interaction diagrams describe the dynamic behavior among different objects. It comprises of a set of objects, their relationships, and the message that the objects send and receive. Thus, an interaction models the behavior of a group of interrelated objects. The two types of interaction diagrams are −

- **Sequence Diagram** − It represents the temporal ordering of messages in a tabular manner.
- **Collaboration Diagram** − It represents the structural organization of objects that send and receive messages through vertices and arcs.

State Transition Diagram

State transition diagrams or state machines describe the dynamic behavior of a single object. It illustrates the sequences of states that an object goes through in its lifetime, the transitions of the states, the events and conditions causing the transition and the responses due to the events.

Concurrency of Events

In a system, two types of concurrency may exist. They are −

System Concurrency

Here, concurrency is modeled in the system level. The overall system is modelled as the aggregation of state machines, where each state machine executes concurrently with others.

Concurrency within an Object

Here, an object can issue concurrent events. An object may have states that are composed of sub-states, and concurrent events may occur in each of the sub-states.

Concepts related to concurrency within an object are as follows –

Simple and Composite States

A simple state has no sub-structure. A state that has simpler states nested inside it is called a composite state. A sub-state is a state that is nested inside another state. It is generally used to reduce the complexity of a state machine. Sub-states can be nested to any number of levels.

Composite states may have either sequential sub-states or concurrent sub-states.

Sequential Sub-states

In sequential sub-states, the control of execution passes from one sub-state to another sub-state one after another in a sequential manner. There is at most one initial state and one final state in these state machines.

The following figure illustrates the concept of sequential sub-states.

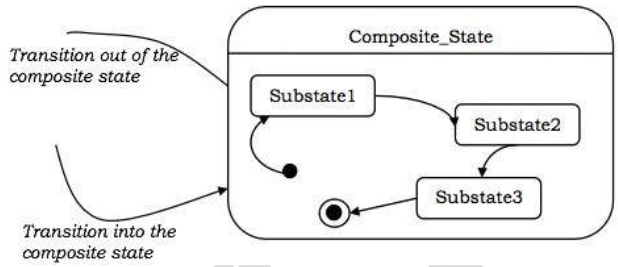


Figure 11-12.3

Concurrent Sub-states

In concurrent sub-states, the sub-states execute in parallel, or in other words, each state has concurrently executing state machines within it. Each of the state machines has its own initial and final states. If one concurrent sub-state reaches its final state before the other, control waits at its final state. When all the nested state machines reach their final states, the sub-states join back to a single flow.

The following figure shows the concept of concurrent sub-states.

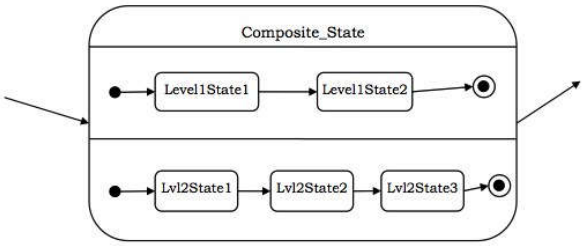


Figure 11-12.4

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). It depicts the 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. 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).

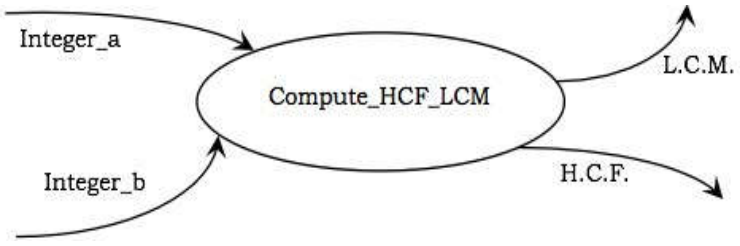


Figure 11-12.5

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, 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 forke

Figure 11-12.6

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.

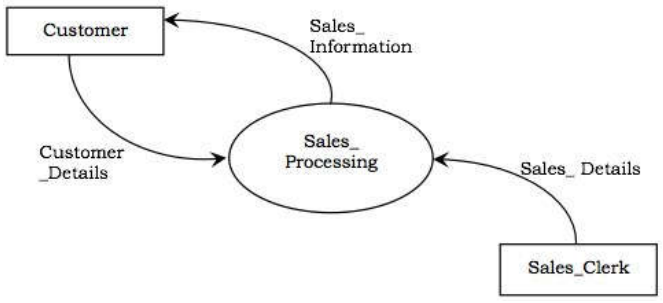


Figure 11-12.7

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 labeled. 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.

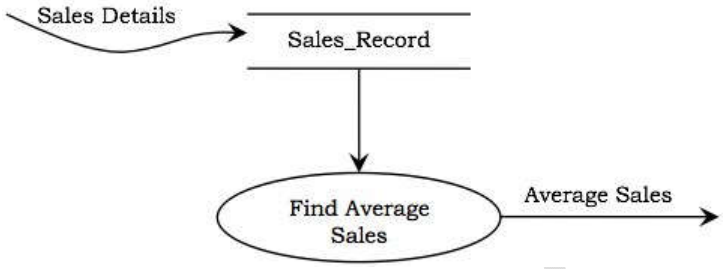


Figure 11-12.8

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 − 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.

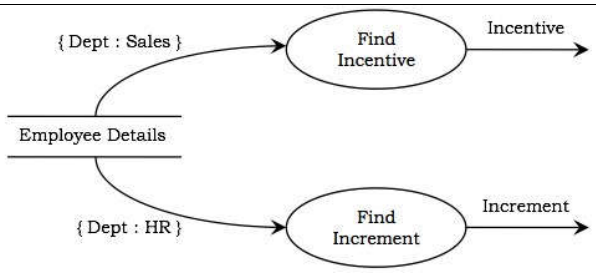


Figure 11-12.9

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.

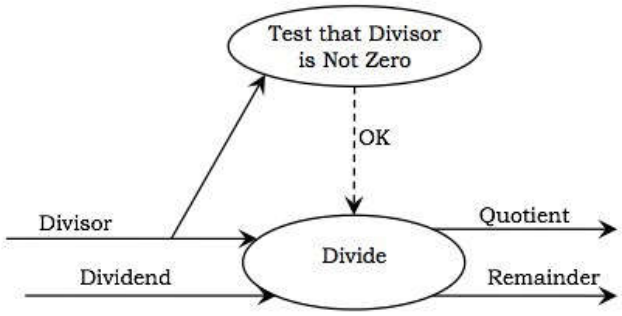


Figure 11-12.10

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

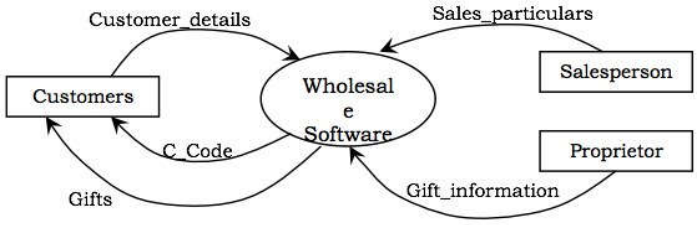


Figure 11-12.11

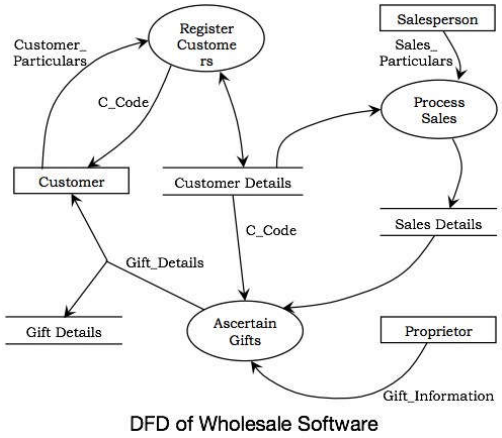
In the next level DFD, as shown in the following figure, the major processes of the system are identified, the data stores are defined and the interaction of the processes with the actors, and the data stores are established.

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

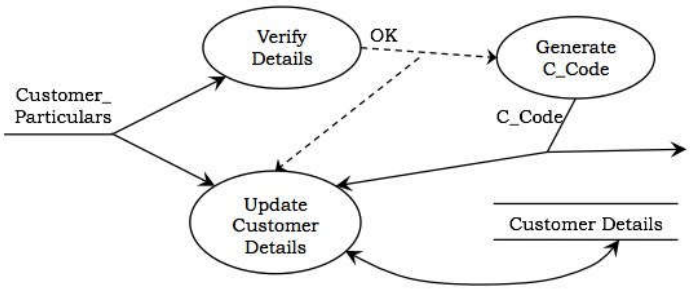


DFD of Wholesale Software

Figure 11-12.12

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

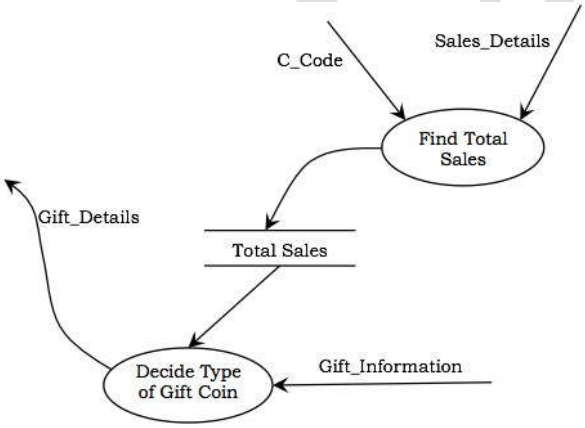
entered, they are verified. If the data is correct, C_Code is generated and the data store Customer Details is updated.



DFD of Customer Process

Figure 11-12.13

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

Figure 11-12.14

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 modeling develops the static structure of the software system in terms of objects. Thus it shows the "doers" of a system.
- Dynamic Modeling 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.

Read Additional Resources

1. **OOAD**
   Week 11-12_Object Oriented Analysis and Design.pdf

Watch Video Resources

1. **Object Oriented Analysis and Design**
   Week 11-12_Object Oriented Programming Fundamental.mp4
2. **Functional Modeling**
   Week 11-12_Functional Modeling.mp4
3. **Component Diagram**
   Week 11-12_Component Diagram.mp4
4. **Class Diagram**
   Week 11-12_Class Diagram.mp4
5. **Analysis Model**
   Week 11-12_Analysis Model.mp4
6. **All About Data Flow Diagrams**
   Week 11-12_Analysis Model.mp4

Self-Check

Quiz 11-12.1

Instructions: Write your answer on the Answer Sheet (AS) provided in this module.

1. What are the major elements of Java?
2. What are the stages of Object Oriented Analysis?
3. What are the different parts of state transactions?

Internet References

1. https://www.youtube.com
2. https://www.tutorialspoint.com
3. www.javatpoint.com
4. www.javatutorials.com