UNIT 2 OBJECT ORIENTED ANALYSIS

Structure **Page Nos.**

2.0 INTRODUCTION

Object oriented analysis (OOA) is concerned with developing software engineering requirements and specifications that expressed as a system's object model composed of a population of interacting objects. The concept of abstraction in object oriented analysis (OOA) is important. Abstraction may be defined as: the characteristics of an object which make it unique and reflect an important concept. Analysis is a broad term, best qualified, as in *requirements analysis* (an investigation of the requirements) or *object oriented analysis* (an investigation of the objects of the problem domain).

OOA views the world as objects with data structures and behaviors and events that trigger operations, for object behavior changes. The idea is to see system as a population of interacting objects, each of which is an atomic bundle of data and functionality, (the foundation of object technology) and provides an attractive alternative for the development of complex systems.

The problem statement is important for any analysis. It is a general description of the user's difficulties, and desires. The purpose of problem statement is to identify the general domain in which you will be working.

In this Unit you will study various analysis techniques: object modeling, dynamic modeling and functional modeling. You will also learn how add operations in system and how to do refining of the analysis model.

2.1 OBJECTIVES

After going through this unit, you should be able to:

- define the concepts of the objects in the system;
- express required system behaviour in terms of business objects in the system, and actions that the user can perform on them;
- understand how to define and analyze the problem statement;
- explain the purpose of object modeling;
- explain dynamic modeling;
- describe the event flow between objects and how to draw state diagrams of real world problems;
- explain and understand the importance of functional model;
- explain how operations can be added in various analysis techniques, and
- explain the importance of iterating or refining the analysis model.

2.2 OBJECT ORIENTED ANALYSIS

In this section we will discuss basics of object oriented analysis with the help of object oriented features.

Analysis

Analysis is not a solution of the problem. Let us see what actually object oriented (OO) analysis. Analysis emphasizes an ***investigation*** of the **problem** and **requirements,** for example, if a new online trading system is desired by a trader, there may be questions such as, how will it be used? What are its functions?

If you start a project the first question about that project' will be how do we get started? Where do we begin?

The starting point for object oriented analysis is to identify candidate objects and their relationships.

The first stage can be a simple brain-storming of the possible objects. Then one method is to go through all the nouns in any documentation about the world you are analysing, and considering these as candidate objects of the system. Also, you can use the alternative technologies for developing software engineering requirements and specifications that include functional decomposition, essential systems analysis, and structured analysis.

The Process of Development

The approach to development can be an iterative one. It involves repeated refinement of the object model. The process needs to be controlled by an appropriate project management process, involving reviews and check pointing at the levels of:

- Analysis
- Design
- Implementation
- Post Implementation Review

Object Orientation and Analysis

An ***Object*** is something that exists within the problem domain that can be identified by data and/or behaviour. An example of an object is a car. The data of a car could be the wheel, brake, seat, etc. The behaviour of the car would be to drive on roads, its speed, etc.

Object oriented analysis is the concept which actually forces you to think in terms of the application domain when its behaviour and data known to you.

In OOA the primary focus on identifying objects from the application domain, then fitting procedures around them.

For example, in the case of the flight information system, the objects would include *Plane, Flight,* and *Pilot,* etc.

The object model has many aspects, which are associated with OO concepts.
Now we will discuss the following principle of OO.

**Abstraction, Encapsulation, Identity, Modularity, Hierarchy,
Typing, Concurrency, and Persistence**

Abstraction

You understand the term object. Now, let us see how problems are seen as objects, and find their associated data and behaviour. You will notice that an object is a real life entity, or abstraction.

In our daily life we deal with complexity by abstracting details away.

Let us see this with an example of:

Driving a car does not require knowledge of internal combustion engine. It is sufficient to think of a car as simple transport.

In simple term, abstraction means to focus on the essential, inherent aspects of an entity, ignoring its accidental properties. Abstraction is a normal process that we do everyday. When you view the world, you can not possibly take in every minute detail, so you concentrate on the aspects that are important.

An abstraction is a simplified description of a system that captures the essential elements of that system (from the perspective of the needs of the modeler), while suppressing all other elements.

Encapsulation

This separates the interface of an abstraction from its implementation. Taking the above example of a car, we can now categorize as:

Abstraction	Car
Interface	Steering, pedals, controls
Implementation	Generally, you don't know

Encapsulation also means **data hiding**, which consists of separating the external aspects of an object, which are accessible to other objects.

Now let us take a example of the Stack.

A Stack abstraction provides methods like push (), pop (), isEmpty(), isFull(). The Stack can be implemented as a singly linked list, or a doubly linked list, or an array, or a binary search tree. This feature is called encapsulation. It hides the details of the implementation of an object.

The Benefits of Encapsulation

To hide the details of a class, you can declare that data or implementation in its private part so that any other class clients will not be able to know about. It will have the ability to change the representation of an abstraction (data structures, algorithms) without disturbing any of its clients. The major benefit of encapsulation is that you.

By Encapsulation

You can delay the resolution of the details until after the design.
Keep your code modular.

Object Identity

Object identity is a property of an object that distinguishes the objects from all other objects in the applications. With object identity, objects can contain, or refer to other objects. You can create an object identity in three ways:

1) You can refer as memory address in programming languages.

2) Assign identifier keys in the database.

3) By user-specified names, used for both programming and database.

In a complete object oriented system each object is given an identity that will be permanently associated with the object irrespective of the object's structural or state transitions. The identity of an object is also independent of the location, or address of the object. Object identity provides the most natural modeling primitive to allow the "same object to be a sub-object of multiple parent objects".

Modularity

Modularity is closely tied to encapsulation; you may think of modularity as a way of mapping encapsulated abstractions into real, physical modules. It is a property of a system that has been decomposed into cohesive and loosely coupled modules.

Cohesion and coupling gives two goals for defining modules. You should make a module cohesive (shared data structures, similar classes) with an interface that allows for minimal inter-module coupling.

It is important to remember that the decisions concerning modularity are more physical issues, whereas the encapsulation of abstractions is logical issues of design.

Hierarchy

This is ranking or ordering of abstraction.
Hierarchy is decided by using the principle of 'divide and conquer'. You can describe complex objects in terms of simpler objects. This is the property of object oriented analysis (OOA) which enables you to *reuse the code.*

You can place an existing class directly inside a new class. The new class can be made up of any number and type of other objects, in any combination that is needed to achieve the desired functionality. This concept is called *composition* (or more generally, *aggregation*). Composition is often referred to as *a "has-a"* relationship *or "part-of"* relationship, for example, in "automobile has an engine" or "engine is part of the automobile."

Typing

This enforces object class such that objects of different classes may not be interchanged. In other words, class can be visualized as a type. Remember that there are two kinds of typing. This typing does not mean the way you usually type the letters.

Strong Typing: When any operation upon an object (which is defined) can be checked at compile time, i.e., type is confirmed forcefully.

Weak Typing: Here, operations on any object can be performed, and you can send any message to any class. Type confirmation is not essential, but in these type of language more errors at execution time may occur.

Concurrency

The fundamental concept in computer programming is the idea of handling more than one task at a time. Many programming problems require that the program be able to:

1) Stop what it's doing

2) Currently deal with some other problem, and

3) Return to the main process. There is a large class of problems in which you have to partition the problem into separately running pieces so that the whole program can be more responsive. Within a program, these separately running pieces are called threads, and the general concept is called *multithreading.*

Currently, if you have more than one thread running that is expecting to access the same resource then there is a problem. To avoid this problem, a thread locks a resource, completes its task, and then releases the lock so that someone else can use the resource. It can lock the memory of any object so that only one thread can use it at a time. It is important to handle concurrent running programs/threads properly.

Persistance

When you create an object, it exists for as long as you need it, but under no circumstances do object exist when the program terminates. While this makes sense at first, there are situations in which it would be incredibly useful if an object could exist

and hold its information even while the program is not running. When, next time you start the program, the object would be there and it would have the same information it had the previous time the program was running. Of course, you can get a similar effect by writing the information to a file or to a database, but in the spirit of making everything an object it would be quite convenient to be able to declare an object persistent and have all the details taken care of for you.

2.3 PROBLEM STATEMENT: AN EXAMPLE

You must understand here that you are looking for a statement of needs, not a proposal for a solution. OOA specifies the structure and the behaviour of the object, that comprise the requirements of that specific object. Different types of models are required to specify the requirements of the objects. These object models contain the definition of objects in the system, which includes: the object name, the object attributes, and the objects relationships to other objects.

As you know, an object is a representation of a real-life entity, or an abstraction. For example, objects in a flight reservation system might include: an airplane, an airline flight, an icon on a screen, or even a full screen with which a travel agent interacts. The behaviour, or state model describes the behavior of the objects in terms of the states of the object and the transitions allowed between objects, and the events that cause objects to change states.

These models can be created and maintained using CASE tools that support the representation of objects and object behaviour.

You can say that the problem statement is a general description of the user's difficulties, desires, and its purpose is to identify the general domain in which you will be working, and give some idea of why you are doing OOA.

It is important for an analyst to separate the true requirements from design and implementation decisions.

Problem statement should not be incomplete, ambiguous, and inconsistent. Try to state the requirements precisely, and to the point. Do not make it cumbersome.

Some requirements seem reasonable but do not work. These kind of requirements should be identified. The purpose of the subsequent analysis is to fully understand the problem and its implications, and to bring out the true intent of the Client.

Λ Check Your Progress 1

1) Give two benefits of Reuse of Code.

……

…..……...

.....

2) Give an example of enforcement in Typing.

……

…..……...

……

3) What are the benefits of OOA technology?

……

…..……..

……

4) Briefly explain what is to be done while defining the problem statement.

……

…..……...…

……

You are already familiar with structured analysis. Now, this is the appropriate time to have a comparison of OOA and Structured analysis.

2.4 DIFFERENCES BETWEEN STRUCTURED ANALYSIS AND OBJECT ORIENTED ANALYSIS

Object oriented analysis design (OOAD) is basically a bottom up approach which supports viewing the system as a set of components (objects) that can be logically kept together to form the system.

Advantages and Disadvantages of Object Oriented Analysis and Design

Advantages:

The OO approach inherently makes each object a stand alone component that can be reused not only within a specific stat problem domain, but also is completely different problem domains, having the requirement of similar objects.

The other main advantage of object oriented (OO) is the focus on data relationships. You cannot develop a successful system where data relationships are not well understood. An OO model provides all of the insight of an ER diagram and contains additional information related to the methods to be performed on the data. We will have more detailed discussion on this aspects in Block 3 of this Course.

Disadvantages:

You know that OO methods only build functional models within the objects. There is no place in the methodology to build a complete functional model. While this is not a problem for some applications (e.g., building a software toolset), but for large systems, it can lead to missed requirements. You will see in Unit 3 of this course. “Use cases” addresses this problem, but since all use cases cannot be developed, it is still possible to miss requirements until late in the development cycle.

Another disadvantage of the object oriented analysis design (OOAD) is in system modeling for performance and sizing. The object oriented (OO) models do not easily describe the communications between objects. Indeed, a basic concept of object oriented (OO) is that the object need not know who is invoking it. While this leads to a flexible design, performance modeling cannot be handled easily.

The object oriented (OO) analysis design itself does not provide support for identifying which objects will generate an optimal system design. Specifically, there is no single diagram that shows all of the interfaces between objects. You will study object oriented analysis design (OOAD) diagrams in Unit 3 of this course. As you know, coupling is a major factor in system complexity, not having this information makes architecture component selection a hit or miss proposition.

Advantages and Disadvantages of Structured Analysis

With experience, you will come to know that most customers understand structured methods better than object oriented (OO) methods. Since one of the main reasons of modeling a system is for communication with customers and users, there is an advantage in providing structured models for information exchange with user groups or customers.

In fact, specifications are typically in the form of a simple English language statement of Work and Requirement Specification. Therefore, the system to be built, must be understood in terms of requirements (functions the system must perform), that is why this naturally leads to a structured analysis, at least at the top level. Specifically structured methods (functional decomposition) provide a natural vehicle for discussing, modeling, and deriving the requirements of the system.

The disadvantage with structured methods is that they do not readily support the use of reusable modules. The top down process works well for new development, but does not provide the mechanisms for "designing in" the use of existing components. The top down process of functional decomposition does not lead to a set of requirements which map well to existing components.

When the requirements do not map cleanly, you have two choices: either you do not use the existing components, or force fit the requirements to the existing components and "somehow" deal with the requirements which are only partially covered by the existing components, which does not lead to a good successful system. Now, we will discuss how actually object oriented analysis (OOA) system is performed.

2.5 ANALYSIS TECHNIQUES

In this section we will see how classes re identified, and their applications with the help of basic modeling techniques.

2.5.1 Object Modeling

Object modelling is very important for any object oriented development, object modeling shows the static data structure of the real world system. Basically, object modeling means identifying objects and classes of a system, or you can say that it describes real world object classes and their relationships to each other. You can develop an object model by getting information for the object model from the problem statement, expert knowledge of the application domain, and general knowledge of the real world. Object model diagrams are used to make easy and useful communications between computer professionals and application domain experts.

To develop an object model first identify the classes and their associations as they affect the overall problem structure and approach. Then prepare a data dictionary.

i) Identify associations between objects.
ii) Identify attributes of objects and links.
iii) Organise and simplify object classes using inheritances.

Then you verify that access paths exist for likely queries

iv) Iterate and refine the model, and
v) Group classes into modules.

Identifying Object Classes

You should always be careful while identifying relevant object classes from the application domain. Objects include physical entities in a system like buildings, employees, department, etc. Classes must make sense in the application domain. At this stage you should avoid computer implementation constructs, such as linked lists and subroutines.

Some classes are implicit in the application domain, so you need to find out by understanding the problem well.

Action-object matrix: A matrix showing how update actions affect objects. It may be considered to be part of the user object model, as it summarizes user object action definitions in a tabular view.

Process of this whole activity is like:

Check for multiple models

- Identify objects
- Create user object model diagram
- Define user object attributes
- Define user object actions
- Create action-object matrix
- Check for dynamic behavior
- Review glossary.

Some notations for user object model:

Notation Meaning

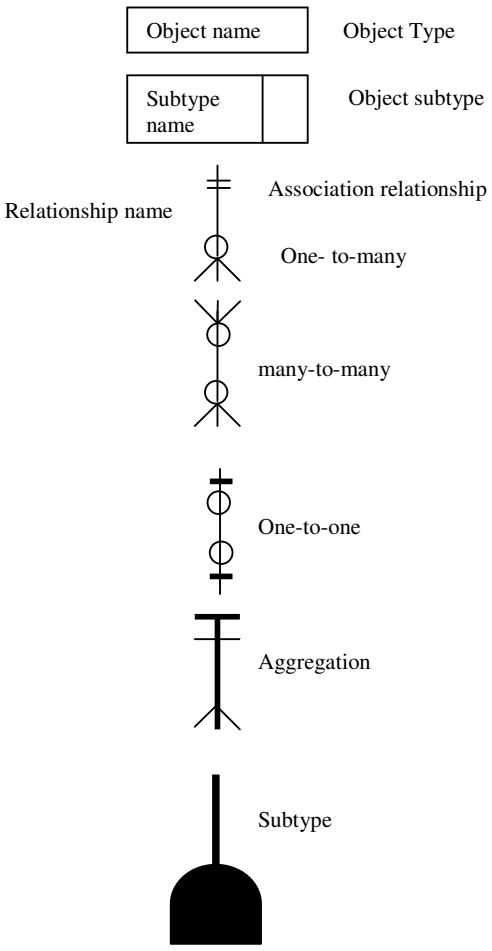

Figure 1: User Object Model Notations

Let us see the example in which we will discuss how concepts are implemented:

To illustrate the object modeling technique to you by taking the very common example of the tasks associated with using a telephone set:

Answer the “phone	
Hear the “phone ringing.	[Ringer]
Identify the ringing “phone.	[Receiver Unit]
Pick up the handset.	[Handset]
Talk to the caller.	[Other party]
Replace the handset.	[Handset]
Make a ‘phone call.	
Pick up the handset.	[Handset]
Listen for the dialing tone.	[Exchange]

'Dial' the number.	[Dial]
Listen for the ringing tone.	[Receiver Unit]
Talk to the person answering.	[Other party]
Replace the handset.	[Handset]

On the basis of above information, the **objects/actions** that may be identified from this are:

Receiver Unit
Identify
Ringer
Hear
Handset
Pick up
Replace
Other party
Talk with
Exchange
Listen to
Dial
Enter number

The attributes of these identified objects are identified as:

Receiver Unit
Identity (phone number)
Status (ringing, engaged)
Ringer
Ringing (true, false)
Handset
Hook (on, off)
Other party
Status (caller, answered)
Exchange
Tone (dead, dialing, other)
Dial
Status (empty, number entered)

You can see the User Object Model diagram produced from this given in *Figure 2.*

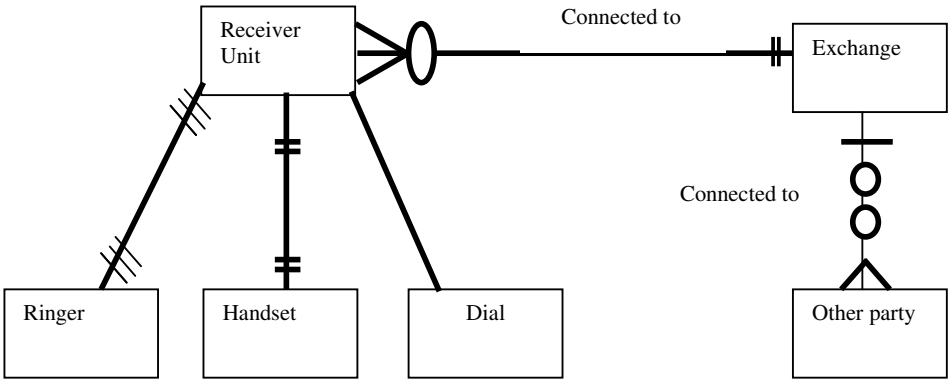

Figure 2: User object Model for Telephone set

2.5.2 Dynamic Modeling

You know that computer systems are built from objects which respond to events. External events arrive at the boundary of the system; you understand the concept of events.

For example, whenever you click with your mouse on the desktop or the canvas area some action is triggered (you get some response back).

In Dynamic modeling you examine "**a way of describing how an individual object responds to events, either internal events triggered by other objects or external events triggered by the outside world**".

Dynamic modeling is elaborated further by adding the concept of time: new attributes are computed as a function of attribute changes over time.

Before you define the dynamic behaviour of user objects. You should know the following:

The Inputs are:

User object model which comprises objects, attributes, actions and relationships.
Task model: from which significant object states may be identified.

Products

Dynamic model: A model of the dynamic behaviour of a user object. It defines significant states of the user object, the way that actions depend on the state, and affect the state.

The dynamic model consists of a dynamic model diagram, showing states and transitions and supplementary notes, specifying states and actions in more detail.

Process of Dynamic modeling:

- Analyse applicability of actions
- Identify object states
- Draw dynamic model diagram
- Express each state in terms of object attributes
- Validate dynamic model
- Concepts.

Dynamic modeling: state diagrams

We will study this in detail in Block 3 of this course.

Generally, a state diagrams allows you to further explore the operations and attributes that need to be defined for an object. They consist of sets of states which an object is in, and events which take the object from one state to another.

State: An object may have one or more states–stable points in its life, expressed by the object's attributes and relationships.

Event/action: Something that happens to an object. Atomic, in that it either has happened or it hasn't. An event causes an action.

Transition:

A jump between states, labeled with the corresponding action.

Notations for state diagram are shown below in *Figure 3*:

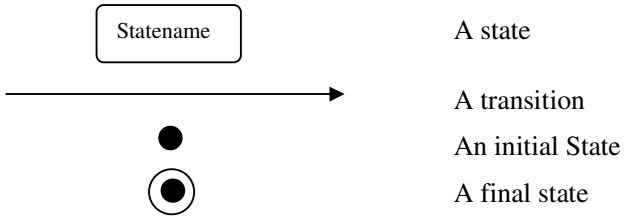

Figure 3: Notation for State Diagram

A simple example using these notation is shown below in Figure 4.

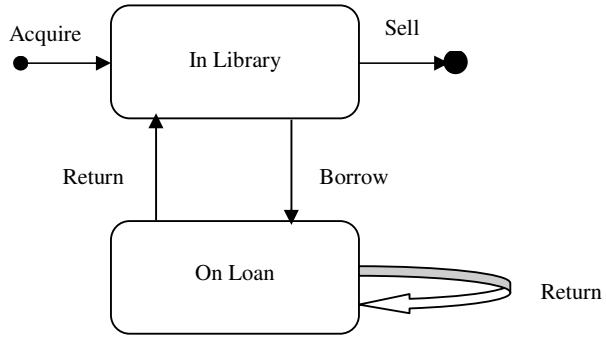

Figure 4: A simple State Diagram

States may be seen as composite states: a collection of states treated as one state at a higher level on analysis. In the example above, the state "In Library" is actually composed of a sequence of sub-states which the library staff, if not borrowers would need to know about.

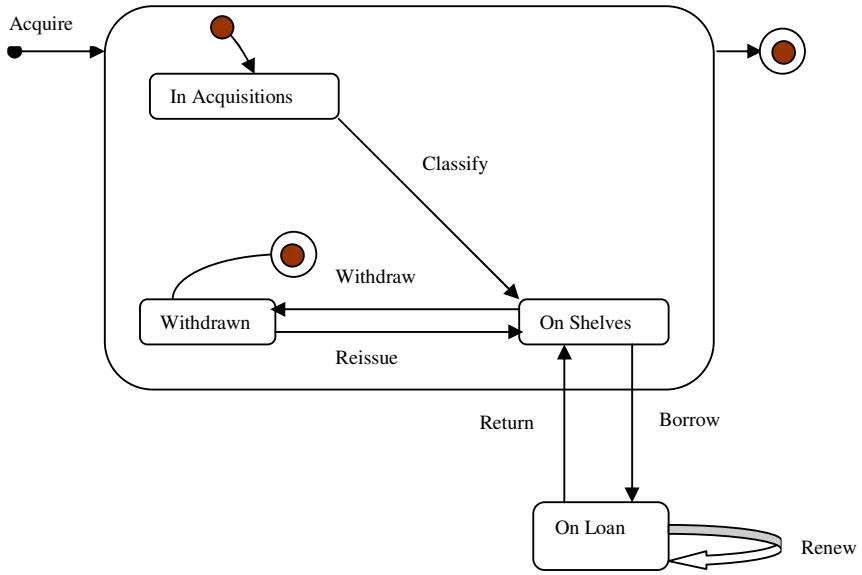

Figure 5: Composite States

Basically any system which you can reasonably model on a computer jumps from state to state.

The level of state decomposition must be decided by judgement. A too fine grained model is inappropriate; for example, modeling all the possible ages of an employee as individual states. Also, a gross decomposition is useless; for example, modeling an employee as employed or not.

You can ask whether all objects of world have behaviour which can be modeled.

Of course, not all objects have behavior worth modeling. Consider our example from the field of object oriented nursery rhymes, the sad story of Humpty Dumpty. Clearly,

you would not like to model such trivial examples.

Basically you need to construct a state transition diagram for every object with significant behaviour. You need not construct one for anything with trivial behaviour.

The reason for doing this is to identify further operations and attributes, to check the logical consistency of the object and to more clearly specify its behaviour.

All the best notations can be used to describe the process they facilitate.

Now we have the basic ideas of object models and dynamic models, our approach to analysis and design (so far) can be summarised as:

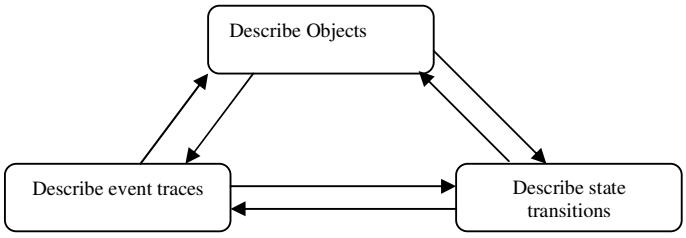

Figure 6: Objects and State transitions

Note: You should also match the events between state diagrams to verify consistency. Till so far, you have been introduced to object modeling, and dynamic modeling. Now, we will see what Function modeling is:

2.5.3 Functional Modeling

You know that Data flow modeling is a common technique used for the analysis of a problem in software engineering. It is exceptionally useful for analyzing and describing systems where there is a large amount of calculation involved.

Data flow models consist of a number of processes which exchange information. A process transforms information it receives, and passes the transformed information on to other processes or to objects in the system. Data flow models can be used to uncover new operations and new attributes for the object model. Sometimes, new objects can be discovered too.

Basically you can state that the functional model shows how values are computed. It describes the decisions or object structure without the regard for sequencing. It gives you dependency between the various data and the functions that relate them, giving the flow of data.

Each process needs to be implemented as an operation in one or more of the objects. Each data item arising from an object must have a corresponding attribute, or set of attributes in the source object. The data flow diagram (DFD) corresponds to activities or actions in the state diagrams of the classes.

That is why it is suggested to construct the functional model after the object and dynamic models.

Now, let us discuss about the creation of the DFD and the functional model.

The approach to data flow diagramming should be as follows:

- Create a data flow diagram for each of the major outputs of the system
- Work back from the outputs to the inputs to construct the diagram
- Add new objects where necessary to the object model as you discover the need for them in the data flow modeling add new operations and attributes to the object model as you discover the need for them in the data flow modeling.

One thing you have to remember is that the data flow diagram is not used as a basis for devising the structure of the system.

Steps in constructing a Functional Model

- Identify input and output values
- Build data flow diagrams showing functional dependencies
- Describe functions
- Identify constraints
- Specify optimisation criteria.

Identifying Input and Output Values

First, identify what data is going to be used as input to the system, and what will be the output from the system. Input and output values are parameters of events between the system and the outside world. You must note that Input events that only affect the flow of control, such as cancel, terminate, or continue. They do not supply input values. For example, supplier code, name, product description, rate per unit, etc. are the inputs to a sales system.

Build data Flow diagrams showing functional dependencies

Data flow diagrams are useful for showing the functional dependencies. In data flow diagrams processes are drawn as bubbles, each bubble containing with the name of the process inside. Arrowhead lines are used to connect processes to each other, and to objects in the system. The lines are label with the information that is being passed. Objects are drawn as rectangular boxes, just as in the object model, but usually with just the name of these objects and not the attributes and operations.

i)

Sales
Quantity delivered
Calculate quantity delivered to customer

iii)

Product
name price vat rate

ii)

Customer
Product line discount quantity discount
Calculate product discount calculate quantity discount

iv)

Invoice
Calculate line item discount Calculate total Deduct quantity discount Print

Figure 7: Some objects in figure 7: a simple DFD is given for sales system

Let us look at a simple example:

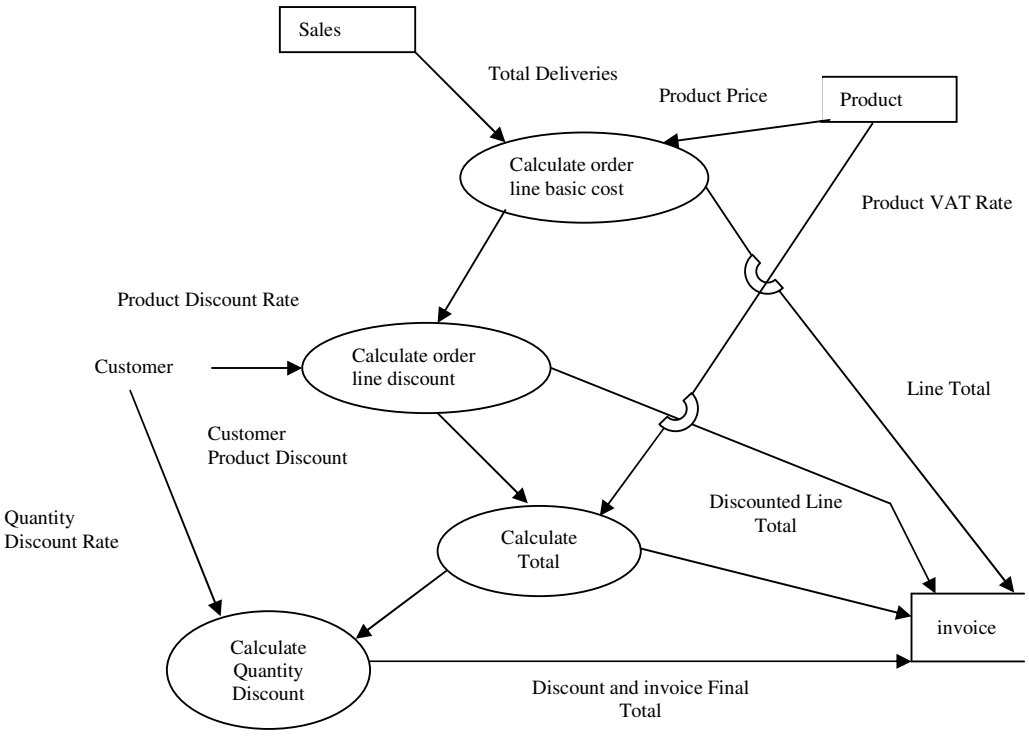

Figure 8: DFD for Sales System

The next stage is to devise operations and attributes to support the calculations.

Describe functions

After you have roughly designed the data flow diagram, you can write a description of each function, and you can describe the function in any form such as, mathematical equations, pseudo code, decision tables, or some other appropriate form. You need not know the implementation of the function, but what it does. The description can be declarative or procedural.

A declarative description specifies the relationship between the input and output values, and relationship among the output values.

A procedural description specifies a function by giving an algorithm to compute it. The purpose of the algorithm is only to specify what the function does.

Identifying Constraints Between Objects

In a system, there are some boundaries to work on. Boundaries or Constraints are the functional dependencies between objects which are not related by an input-output dependency. You have to find out the constraints between the objects. Constraint has different behavior at different times. It can be on two objects at the same time, between instances of the same object at different times (an invariant), or between instances of different objects at different times.

You can define constraints as Preconditions (input values) and PostConditions (output values). Preconditions on functions are constraints that the input values must satisfy. Post conditions are constraints that the output values must satisfy to hold. State the conditions under which constraints hold.

Specifying Optimisation Criteria

Specify values to be maximized, minimized, or optimized. You can understand it as the way you normalize the data in the database. For example, you should minimize the time an account is locked for concurrency reasons. If locks are needed, then it is extremely important to minimize the time that an entire bank is locked for concurrency reasons.

☞ **Check Your Progress 2**

1) Explain how you can define an object model of a system.

……

…..……..…

……...

2) Show the basic dynamic model of telephone.

……

…..……

2.6 ADDING OPERATIONS

Whenever you look at the operations in OOPs you find queries about attributes or associations in the object model (such as student.name), to events in the dynamic model (such as ok, cancel), and to functions in the functional model (such as update, save).

You can see operations from the object model. It includes reading and writing attribute values and association links. In the object model operations are presented with an attribute. During analysis nothing is private, you assume that all attributes are accessible.

Accessing from one object to another through the object model can be referred to as "pseudo-attribute" like Account.Bank, where account and bank are two separate objects of their respective classes.

Operations from events

During analysis, events which are sent to the target objects. An operation on the object are presented as labels on transitions and should not be explicitly listed in the object model.

Events can be expressed as explicit methods.

You can also implement events by including event handler as part of the system substrate.

Operations from State Action and Activities

You must see that State actions and activities are actually functions, which can be defined as the operations on the object model.

Operations from Functions

As you know, function are actually operations on object. These functions should be simple and summarized on the object model. Organise the functions into operations on objects. For example, the select operations are really path traversals in the object model. The operations like withdrawal-money, verify-password are the operations on class Account of Bank Management system.

You can write as:
account: withdraw (code, amount)->status
account: deposit (code, amount)->status.

☞ Check Your Progress 3

1) Describe how you can simplify Operations.

……

……..….

……

2) Give an example of operations from State Actions and Activities.

……

…..……...

……

3) Give an example of Operations from functions of a bank.

……

…..……..

……

4) How do you see the final model after iterative analysis?

……

…..……...

……

5) Why is iterative analysis of any problem needed?

……

…..……..

……

Iteration of analysis is important. Let us see how OOA apply iteration

2.7 ANALYSIS ITERATION

To understand any problem properly you have to repeat the task which implies that analysis requires repetition. First, just get the overview of the problem, make a rough draft, and then, iterate the analysis according to your understanding. Finally, you should verify the final analysis with the client or application domain experts. During the iteration processes refining analysis and restating of requirement takes place.

2.7.1 Refining the Ratio Analysis

Basically, refinement leads to purity. So to get a cleaner, more understandable and coherent design, you need to iterate the analysis process.

- Reexamine the model to remove inconsistencies, and imbalances with and across the model.
- Remove misfit, and wrong concepts from the model.
- Refining sometimes involves restructuring the model**.**
- Define constraints in the system in a better way.
- You should keep track of generalizations factored on the wrong attributes.
- Include exceptions in the model, many special cases, lack of expected symmetry, and an object with two or more sets of unrelated attributes, or operations.
- You should remove extra objects or associations from the model. Also, take care to remove redundancy of objects and their extra attributes.

2.7.2 Restating the Requirements

To have clarity of the analytical model of the system you should state the requirements specific performance constraints with the optimization criteria in one document verify in the other document. You can state the method of a solution.

Verify the final model with the client. The requirement analysis should be confirmed and clearly understood by the client**.**

The impractical, or incorrect, or hypothetical objects that do not exist in the real world should be removed from the proposed system.

You must note that the analysis model is the effective means of communication with application experts, not computer experts. In summary, the final model serves as the basis for system architecture, design, and implementation.

2.8 SUMMARY

In this Unit you have learned that the goal of analysis is to understand the problem and the application domain, so that you come out with a well cohesive design. There is no end or border line between the analysis and the design phase in software engineering. There are three objectives of the analysis model:

Object oriented analysis (OOA) describes what the customer requires, it establishes as basis for the creation of a software design, and it defines a set of requirements that can be validated.

You also remember that the requirement analysis should be designed in such a way that it should tell you what to be done, not how it is implemented.

The object model is the principal output of an analysis and design process.

Dynamic modeling is elaborated further by adding the concept of time: new attributes are computed as a function of attribute changes over time. In this unit, after defining the **scenario** of **typical** and **exceptional sessions**, identify events followed by the building of state diagrams for each active object showing the patterns of events it

receives and sends, together with actions that it performs. You should also match the events between state diagrams to verify consistency.

You have learned that a functional model shows how values are computed; it describes the decisions, or object structure without regard for sequencing.

It gives you dependency between the various data and the functions that relate them, giving the flow of data. Here you construct the data flow diagram which interacts with internal objects and serves as, data stores between iterations. You also specify the required constraints, and the optimisation criteria.

You have seen that refining and restating will give you more clarity of the analytical model of the system.

2.9 SOLUTIONS / ANSWERS

Check Your Progress 1

1) **Reusing the implementation**. Place an existing class directly inside a new class. The new class can be made up of any number and type of other objects, in any combination that is needed to achieve the desired functionality. This concept is called *composition* (or more generally, *aggregation*). Composition is often referred to as a "has-a" relationshp or "part-of" relationship, as in "automobile has an engine" or "engine is part of the automobile."

   Reusing the interface. Take an existing class and make modifications or additions to achieve desired functionality. This concept is called *inheritance*. The original class is called the Base class or Parent class and the modified class is called the *Derived* or *Inherited* or *Sub* or *Child* class.

2) You can understand the concept of enforcement as it make sure objects of different classes may not be interchanged as follows:

   Example: Vegetable v; Fruit f; Mango m;

   This implies 'v' is variables of class Vegetable, 'f' of class Fruit and 'm' of class Mango. Typing ensures that value of 'f' cannot be assigned to 'v'. However, if Mango extends Fruits, then 'f' can be assigned a value of 'm'. The variable of a sub class can be assigned to variable of super class, but not the other way around, and 'm' cannot be assigned a value of 'f'.

3) Using the OOA technology can yield many benefits, such as:

   i) Reusability of code
   ii) Productivity is gained gains through direct mapping
   iii) Maintainability, through this simplified mapping to the real world is possible.

4) To define the Problem Statement of a system, define what is to be done, and how you are going to implement that. Your statement should include the mandatory features, as well the optional features of the system to be developed.

   You should include:
   What is the problem and its scope,
   What is needed
   Application context
   Assumptions
   Performance needs.

Check Your Progress 2

1) A list of terms which will be used by end users to describe the state and behaviour of objects in the system.

Different user classes may require different mental models of the objects in the system. This includes:
What type of objects there are (user objects).
What information the user can know about an object of a particular type (user object attributes).
How the objects may be related to other objects (relationships).
Object types with 'subtypes' which have additional specialised actions or attributes, i.e., User object, Container objects, User object action, User object subtype.
A model of the business objects which end users believe interact with in a GUI system.

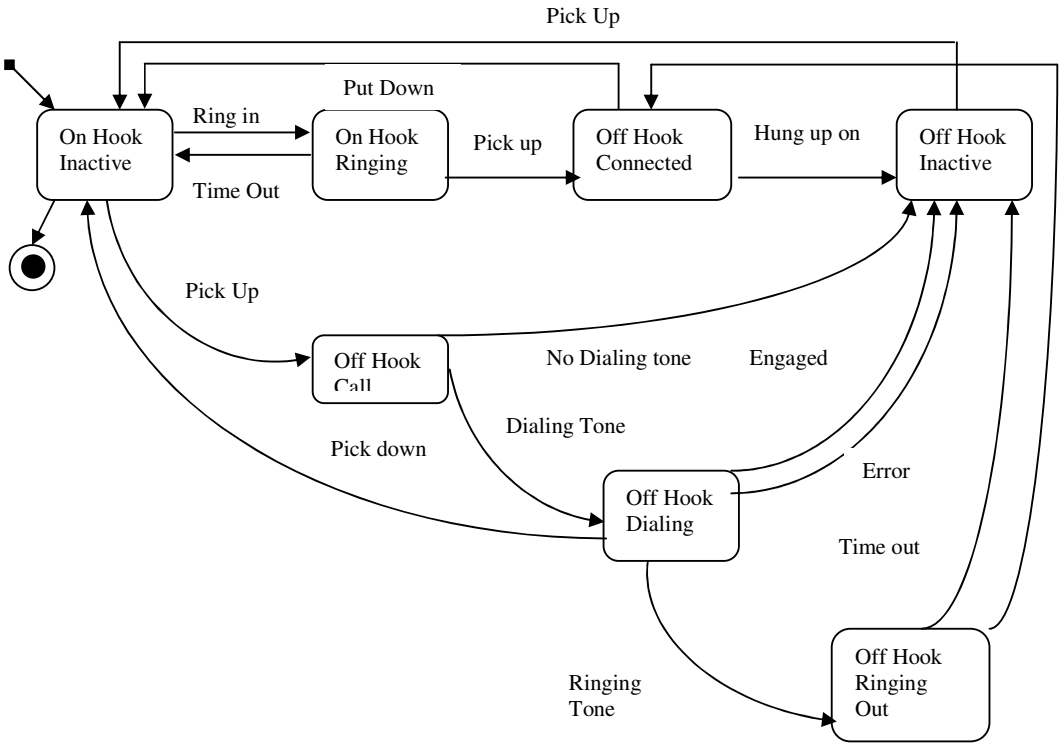

Figure 9: Dynamic Model of Telephone

Check Your Progress 3

1) To simplify the operation, one should use inheritance, where possible, to reduce the number of distinct operations. Introduce new superclasses as needed to simplify the operations, but they should not be forced or unnatural.
Locate each operation at the correct level in the class hierarchy.

2) For example, in the bank the activity, *verify account code* and *verify password*

3) For the creation of a saving account, you may write:
bank::: create –savings-account (customer)->account.
For the operation of checking account, you can write:
bank:: :create-checking-account

4) The final model serves as the basis for system architecture, design, and implementation.

5) The iterative analysis is required to get cleaner, more understandable, and coherent design.