

PREVIOUS YEARS QUESTIONS

PART-A

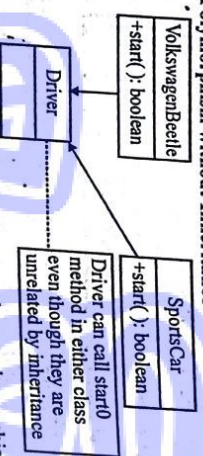
Q1 "Can we have inheritance without polymorphism?"

Comment on it.

[R.T.U. 2018]

Ans. Inheritance and polymorphism are independent but related entities – it is possible to have one without the other. If we use a language that requires variables to have a specific type (C++, C#, Java) then we might believe that these concepts are linked.

Polymorphism without Inheritance



There are languages where you have polymorphism without using inheritance. Some examples are JavaScript, Python, Ruby, VB.NET, and Small Talk.

In each of these languages it is possible to write car.start() without knowing anything about the object car and its method.

```

# Python polymorphism
class VolkswagenBeetle(Car):
    def start(): # Code to start Volkswagen
class SportsCar(Car):
    def start(): # Code to start SportsCar
# Invocation of polymorphism
cars = [VolkswagenBeetle(), SportsCar()]
for car in cars:
    car.start()
    
```

The ability to get pure polymorphism stems from these languages only have a single variable type prior to runtime i.e. var in JavaScript, def in Python, dim in VB.NET. With only one variable type there cannot be type error prior to runtime.

Q2 List object oriented design approaches.

[R.T.U. 2018]

Ans. Object Oriented Design : Object oriented design works around the entities and their characteristics instead of

OBJECT ORIENTED ANALYSIS & DESIGN 5

CHAPTER IN A NUTSHELL

Object Oriented Concepts

Object Oriented Programming is not so much a coding technique as it is a code packaging technique, a way for code suppliers to encapsulate functionality for delivery to customers.

Once the class has been defined, the attributes can be reused when new instances of the class are created.

Following are its basic concepts :

1. Classes & Object
2. Attributes
3. Operations, Methods & Services
4. Messages
5. Encapsulation, Inheritance and Polymorphism.

Object Oriented Analysis Modelling

The analysis model must achieve three primary objectives, which are as follows :

1. To achieve what the customer requires.
2. To establish a basis for the creation of a software design
3. To define a set of requirements that can be validated once the software is built.

The basic element of the analysis model are data dictionary, entity-relationship diagram, Data object description, process specification (PSPEC) & control specification (CSPEC)

Data Modelling

Data Modelling answers a set of specific questions that are relevant to any data processing application. Data Modelling methods make use of the entity relationship diagram (ERD). The ERD enables a software engineer to identify data objects & their relationships using a graphical notation. The entity relationship diagram focuses solely on data, representing a data network that exists for a given system.

The ERD are especially useful for applications in which data & the relationship that govern data are complex.

Object Oriented Design Concepts & Methods

The design of object oriented software requires the definition of a multilayered software architecture, the specification of sub-systems that perform required function & provide infrastructure support, a description of objects that form the building blocks of the system.

The most important early OOD methods are as follows:

- The Booch method
- The Rumbaugh method
- The Jacobson method
- The Coad & Yourdon method
- The Wirfs-Brock method

Class & Object (Definition)

A class is an object-oriented concept that encapsulates the data & procedural abstractions required to describe the content & behaviour of some real world entity.

An object encapsulates both data (attributes) and the functions (operation, methods or services) that manipulate the data.

Refinement

Stepwise refinement is a top-down design strategy originally proposed by Niklaus Wirth-Refinement is actually a process of elaboration. It causes the designer to elaborate on the original statement, providing more & more detail as each successive refinement occurs.

Class & Object Relationships

A relationship exists between any two classes that are connected, the most common type of relationship is binary i.e. a connection exists between two classes. Relationships can be derived by examining the stative verbs or verb phrases in the statement of scope or use cases for the systems.

Human Factors

Human factors can be defined as a discipline that discovers through study & measurement information about human abilities, limitations & others & applies this knowledge to the design of tools, machine, system etc.

Human Computer Interface Design Guidelines

- Place the user in control
- Reduce the user's memory load.
- Make the interface consistent.

Styles of Human Computer Interaction (HCI)

Human Computer interaction is a discipline concerned with the design, valuation & implementation of interactive computing systems for human use & with the study of major phenomena surrounding them.

User interface design :

- Interface Design Models
- Interface Design Process

The above figure shows that customers first order the meal by choosing from menu and then placing orders. Further the customer pays the bill and gets the receipts.

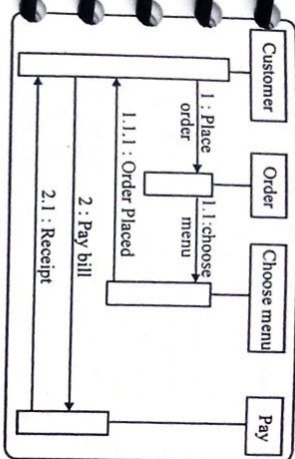


Fig.

2.4 Define class and object in context of UML.

Ans. Classes and Objects : Class is an UML representation which has basic three section.

- (1) Class name
- (2) Class attributes
- (3) Class functions

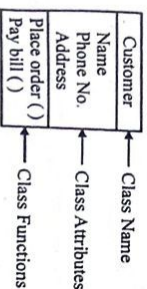


Fig.

Object is an UML representation that is used to represent objects. It has two sections.

- (1) Object name
- (2) Class name

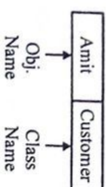


Fig.

Amit is one of the customers.

1.5 Write short note on state diagram.

Ans. State Diagrams : State diagram models the dynamic nature of a system. It describes the various states and the events that changes the state. So, it controls the flow of events from one state to the other.

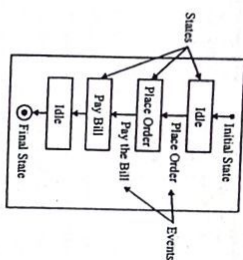


Fig.

Components of state Diagram

- (1) States
- (2) Events triggering to a state.

The above figure shows three states idle, place order and pay bill. Place order is the event that triggers from idle to place order state where if pay the bill event triggers then it leads to pay Bill state.

PART-B

Q.6 Define Unified Approach? Explain UML with all its diagram and advantages.

OR

What is UML? Explain how it is useful in object oriented modeling.

OR

Explain unified modelling language diagrams with the help of appropriate example.

OR

What is UML? How it is useful in OO modelling?

OR

What is UML? Draw and explain class diagram, object diagram and process diagram. List all the diagram available in UML.

OR

Explain different types of diagrams proposed by UML.

OR

What is Unified Modelling Language? Explain all the diagrams of UML?

OR

Ans. Unified Approach : A unified approach evolved from the combined efforts of Booch, Rumbaugh and Jacobson. This unified approach is known as UML.

Software Engineering

UML is organized into two major design activities System design and Object design.

Modelling is simplification of reality. It is a central part of all the activities that leads to the deployment of good software. We build models to communicate the desired structure and behaviour of the system. In short, building models help in:

- (a) Visualization and control of system's architecture.
- (b) Exposure of opportunities for simplification and reuse.
- (c) Managing risk.
- (d) Communication for the desired structure and behaviour of the system.

So, modelling is very important to be done. In the concern of complexity of a program, the modelling plays an important and effective role to reduce it. As we know that when the levels of abstraction increases, there are more chances of failure and it will be very difficult to solve the problem.

The Unified Modelling Language (UML) is a standard language for writing software blueprints. UML may be used to visualize, specify, construct and document the artifacts of a software-intensive system.

Conceptual Model of UML

UML requires three major elements to build its conceptual model. It includes:

- UML's basic building blocks.
- Rules that dictate how those building blocks may be put together and
- Some common mechanisms that apply throughout UML.

One more term required to build UML is view. A view is simply a subset of UML. Modelling construct, that represents one aspect of the system. One or two kinds of diagrams provide a visual notation for the concepts in each view.

UML can be used for domains like :

- Enterprise Information Systems
- Banking and Financial Services
- Telecommunications
- Transportation Management
- Defense/Aerospace and Navigation
- Scientific and Engineering Applications

Building Blocks in UML

To build the blocks in UML for an application, things are defined and the relationships among them are defined. At last complete UML diagram is made.

(A) Things : Things are the abstractions that are present as an entity in a model. There are four kinds of things in UML

1. **Structural Things :** Structural things are the nouns of UML models. They are mostly the static parts of a model. There are seven kinds of structural things :

- Class
- Interface

- Collaboration

- Use-Case
- Active Class
- Component
- Node

2. **Behavioural Things :** Behavioural things are the dynamic parts of UML models. These are the verbs of a model, representing behaviour over time and space. There are two kinds of behavioural things :

- **Interaction :** Interaction is a behaviour that comprises a set of messages exchanged among a specific purpose.

- **State machine :** A state machine shows behaviour particular context to accomplish a specific purpose.

3. **Grouping Things :** Grouping things are the organizational parts of UML models. These are only one into which a model can be decomposed. There is only one kind of grouping things i.e. package. Package is a mechanism for organizing elements into groups.

4. **Annotational Things :** Annotational things are the explanatory parts of UML models. These are the comments applied to describe, illuminate and remark about any element in a model. It should be noted that it is simply a symbol for rendering constraints and comments attached to an element or a collection of elements.

(B) **Relationships :** There are basically four kinds of relationships in UML. They are described below one-by-one.

1. **Dependency :** Dependency is a semantic relationship between two things in which a change to one thing (the independent thing) may affect the semantics of the other thing (the dependent thing). It is shown in fig.1 as a dotted line with an arrow.

Fig. 1 : Dependency

2. **Association :** Association is a structural relationship that describes a set of links, a link being a connection among objects. It is a solid line, possibly directed and may contain label. It may contain multiplicity and role names as shown in fig.2

0..1

Class

Student

Fig. 2 : Association

3. **Generalization :** Generalization is a relationship in which objects of the element that is specialized (may be considered as child) are substitutable for objects of the generalized element (parent). So, a child is sharing the behaviour and structure of parent. It is represented as shown in fig.3. A hollow arrowhead is pointing to the parent.

Fig. 3 : Generalizations

4. **Realization :** Realization is a semantic relationship between classifiers. One classifier specifies a contract that

another classifier guarantees to carry out. Its representation is shown in fig.4

Fig. 4 : Realization

UML Diagrams

Diagrams are the graphical representations of a set of elements. They are represented as a connected graph of vertices (which are things) and arcs (relationships).

So, a diagram is a projection into a system. The same element may appear in all diagrams, in some of the diagrams or in no diagrams at all. A diagram may contain any combination of things and relationships. So, these UML diagrams are the means by which the building blocks can be viewed and these blocks are most often rendered as a connected graph (things) and arcs (relationships).

So UML can be useful in OO modelling as :

- UML defines a number of diagrams so that we can focus on different aspects of the system independently.
- We can have several hundred classes in the design of a corporate enterprise information system. We can never visualize the structure or behaviour of that system by staring at one large diagram containing all the classes and their relationships.

So, if we want to create several diagrams each focused on one view. That's why, different kinds of diagrams are made available in UML to :

- achieve any combination of elements in the same diagram. For example, to depict both classes and objects in the same diagram.
- decide which views we need to best express the architecture of the system and expose the technical risks to the project if any.

The UML includes nine such diagrams :

- Class Diagrams
- Object Diagrams
- Use Case Diagrams
- Sequence Diagrams
- Collaboration Diagrams
- State Chart Diagrams
- Activity Diagrams
- Component Diagrams
- Deployment Diagrams

All these diagrams explained as follows :

1. Class Diagrams : Class diagrams show a set of classes, interfaces, collaborations and their relationships. Class diagrams are specifically used to model the static design view of a system.

Consider an example of house building. During its building, we include basic building blocks such as wall, floors, doors etc. These are inherently structural in nature as they are related with height, width etc. But they are somewhat behavioural also (different kinds of walls can support different

loads). We can't consider these features independently. The process of house construction involves assembling these things.

So, with UML, we use class diagrams to visualize the static aspect of the building blocks and their relationships and to specify their details.

The various contents of a class diagram are :

- Classes
- Interfaces
- Collaborations
- Relationships

The class can be better understood with the help of examples.

Example-1 : In fig.5 a class diagram is shown for the withdrawal of money from the account.

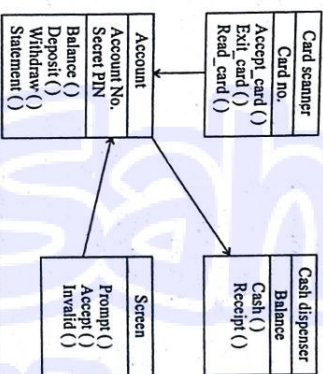


Fig. 5 : Class Diagram for Withdrawal of Money

Example-2 : This example shows the class diagram of an organization, in which department, office, headquarters etc. are shown as things. Its diagram is shown in fig.6

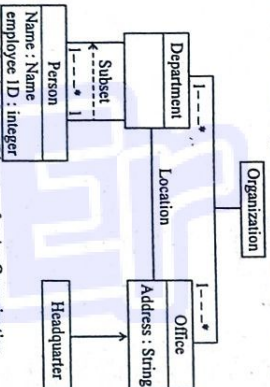


Fig. 6 : Class Diagram for An Organization

2. Object Diagrams

Object diagrams model the instances of things contained in a class diagram, shows set of objects and their relationships at any time and, Modelling a snapshot of system at that moment in time and rendering a set of objects, their state and their relationships.

To understand how the object diagram works, we can visualize a game of soccer. Its overview looks as if it is a

terribly simple sport with a mob of people running wildly chasing a ball. There hardly seems any systematic sense in it.

Now, for a moment, freeze the motion. Classify the individual players. We are able to distinguish their particular positions like forwards, modified, left wing, full back, right back, defenders, goal keepers etc. Further insight reveals how these players collaborate following strategies for attacking and defending. Similar to this, we ought to have the snapshot of the system under concern to look at the objects present, their neighbours and their relationships to these neighbours.

Contents of Object Diagram

There are two contents of an object diagram. They are :

- Links
- Objects

An object diagram expresses the static part of an interaction, consisting of the objects that collaborate but without any of the messages passed among them.

Example-3 : This example tells the object diagram for a company as illustrated in fig.7

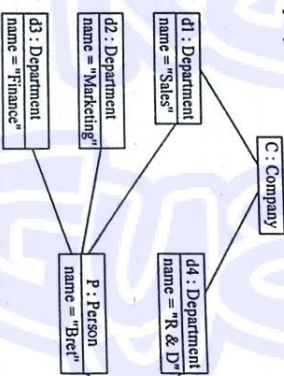
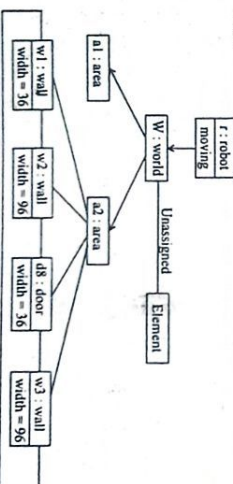


Fig. 7 : Object Diagram for Company

Example-4 : Another example is the objects involved in the mechanism used by the robot to calculate a module of the world in which it moves. Element represents instances that the robot has identified but not yet assigned in its world view.



- **Modelling the requirements of a system** : It involves specifying what that system should do.

Example-5 : Consider the following requirements for a system:

- **Modelling the requirements of a system** : It involves specifying what that system should do.

Example-5 : Consider the following requirements for a system:

cellular phone we identify the elements that form the system. The actors are cellular network and user. And the use cases are making a call, sending message and capturing photos. The use case diagram of cellular telephone is shown in fig. 12

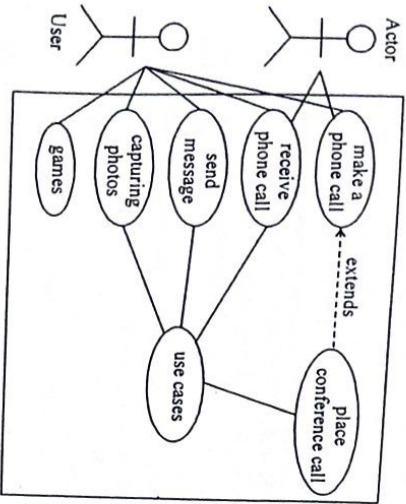


Fig. 12 : Cellular Telephone – Use Case Diagram
Example-6 : Consider an example to process a credit card, using the use cases. It is illustrated in fig. 13

Credit Card Processing

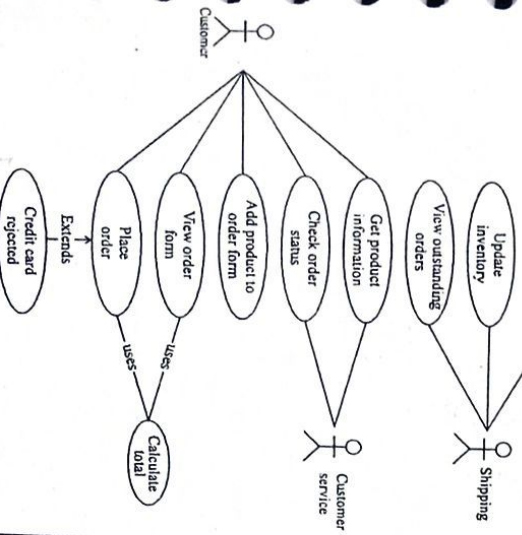


Fig. 13 : Use Case Diagram for Credit Card Processing

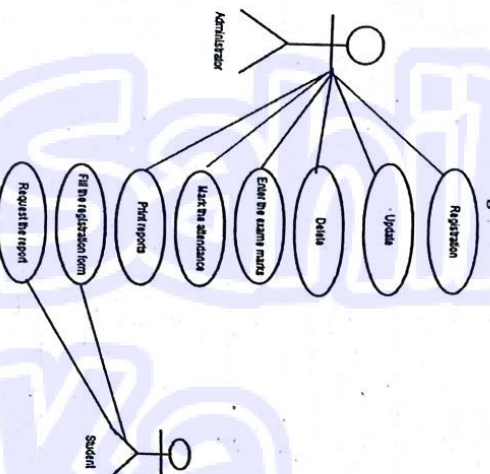
B.Tech. (III Sem.) C.S. Solved Papers

Consider a student Information System as an Example. Explain the following UML diagrams in details, with the help of neat sketches.

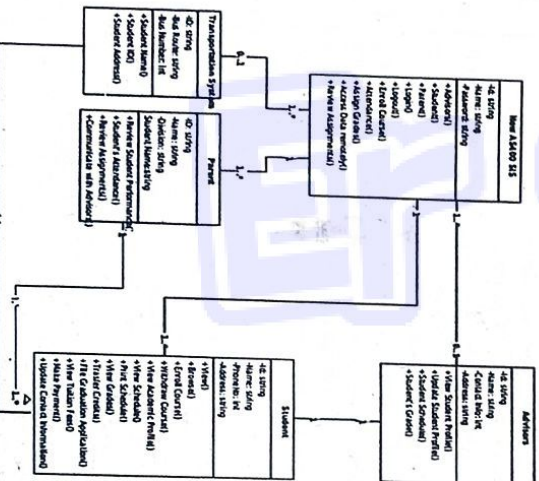
- (a) Use Case Diagram
- (b) Class, Object diagram
- (c) Deployment diagram
- (d) Sequence diagram

[R.T.U. 2016]

Ans.(a) Use Case Diagram



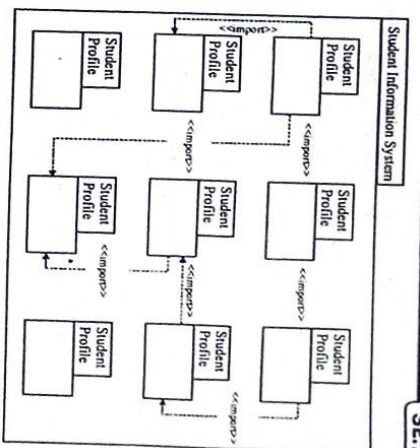
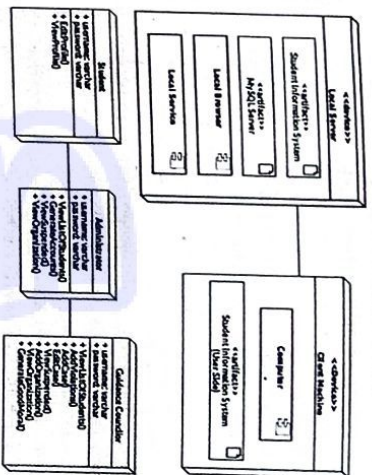
(b) Class, Object diagram



Software Engineering

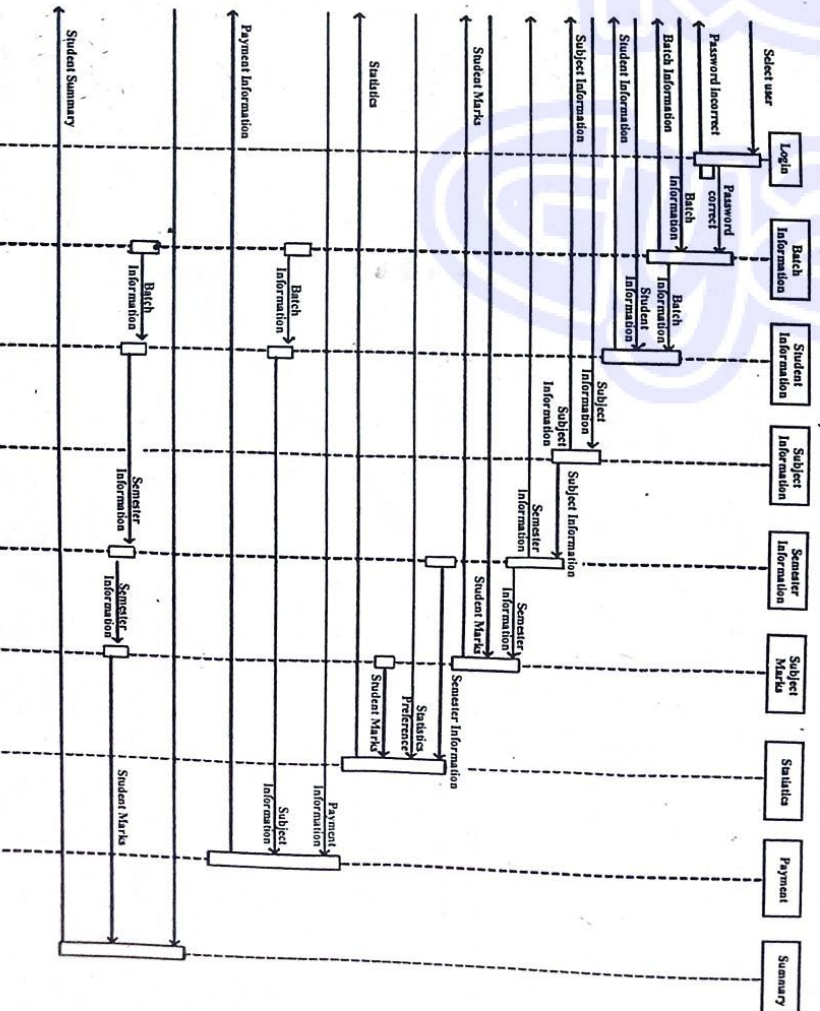
(c) Deployment diagram

Student Information System Deployment Diagram



CE 74

(d) Sequence diagram



2.8 (a) List out the various activities that are encompassed by system design process under object oriented design (OOD) and explain each one briefly.

(b) Explain the object modularization with example.
RTU 2015

Ans.(a) Object-Oriented System Design : Object-oriented system design involves defining the context of a system followed by designing the architecture of the system.

- **Context :** The context of a system has a static and a dynamic part. The static context of the system is designed using a simple block diagram of the whole system which is expanded into a hierarchy of subsystems. The subsystem model is represented by UML packages. The dynamic context describes how the system interacts with its environment. It is modelled using use case diagrams.

- **System Architecture :** The system architecture is designed on the basis of the context of the system in accordance with the principles of architectural design as well as domain knowledge. Typically, a system is partitioned into layers and each layer is decomposed to form the subsystems.

(i) **Object-Oriented Decomposition**

Decomposition means dividing a large complex system into a hierarchy of smaller components with lesser complexities, on the principles of divide-and-conquer. Each major component of the system is called a subsystem. Object-oriented decomposition identifies individual autonomous objects in a system and the communication among these objects.

(ii) **Identifying Concurrency**

Concurrency allows more than one objects to receive events at the same time and more than one activity to be executed simultaneously. Concurrency is identified and represented in the dynamic model. To enable concurrency, each concurrent element is assigned a separate thread of control. If the concurrency is at object level, then two concurrent objects are assigned two different threads of control. If two operations of a single object are concurrent in nature, then that object is split among different threads.

(iii) **Identifying Patterns**

While designing applications, some commonly accepted solutions are adopted for some categories of problems. These are the patterns of design. A pattern can be defined as a

B.Tech. (III Sem.) C.S. Solved Papers

documented set of building blocks that can be used in certain types of application development problems.

Some commonly used design patterns are:

- Facade pattern
- Model view separation pattern
- Observer pattern
- Model view controller pattern
- Publish subscribe pattern
- Proxy pattern

(iv) **Controlling Events**

During system design, the events that may occur in the objects of the system need to be identified and appropriately dealt with.

An event is a specification of a significant occurrence that has a location in time and space.

There are four types of events that can be modelled, namely:

- **Signal Event :** A named object thrown by one object and caught by another object.
- **Call Event :** A synchronous event representing dispatch of an operation.
- **Time Event :** An event representing passage of time.
- **Change Event :** An event representing change in state.

(v) **Handling Boundary Conditions**

The system design phase needs to address the initialization and the termination of the system as a whole as well as each subsystem. The different aspects that are documented are as follows:

- The start-up of the system, i.e., the transition of the system from non-initialized state to steady state.
- The termination of the system, i.e., the closing of all running threads, cleaning up of resources, and the messages to be sent.
- The initial configuration of the system and the reconfiguration of the system when needed.
- Foreseeing failures or undesired termination of the system.

Boundary conditions are modelled using boundary use cases.

(vi) **Object Design**

After the hierarchy of subsystems has been developed, the objects in the system are identified and their details are designed. Here, the designer details out the strategy chosen during the system design. The emphasis shifts from application domain concepts toward computer concepts. The objects identified during analysis are etched out for implementation

Software Engineering

with an aim to minimize execution time, memory consumption, and overall cost.

Object design includes the following phases:

- Object identification
- Object representation, i.e., construction of design models
- Classification of operations
- Algorithm design
- Design of relationships
- Implementation of control for external interactions
- Package classes and associations into modules

(vii) **Design Optimization**

The analysis model captures the logical information about the system, while the design model adds details to support efficient information access. Before a design is implemented, it should be optimized so as to make the implementation more efficient. The aim of optimization is to minimize the cost in terms of time, space, and other metrics.

However, design optimization should not be excess, as ease of implementation, maintainability, and extensibility are also important concerns. It is often seen that a perfectly optimized design is more efficient but less readable and reusable. So the designer must strike a balance between the two.

The various things that may be done for design optimization are:

- Add redundant associations
- Omit non-usable associations
- Optimization of algorithms
- Save derived attributes to avoid re-computation of complex expressions

(viii) **Design Documentation**

Documentation is an essential part of any software development process that records the procedure of making the software. The design decisions need to be documented for any non-trivial software system for transmitting the design to others.

Ans.(b) Its a technique to divide a system into multiple independent and discrete models, which are expected to carry out tasks individually. These may work as basic construct for entire software. Designers tend to design modules such that they can be executed or compiled separately and independently.

Modular design follow rules of, divide and conquer, problem solving strategy. This is because there are many benefits attached to modular design of software.

These are interchangeable modules, such that each contain everything necessary to execute only one aspect of

the desired functionality. Thus, these modules performs logically discrete functions. These properties make modular design system, if built correctly, for more reusable then traditional monolithic design, since all these modules may be reused in other projects.

A simple diagram can be drawn to explain the concept Ubiquitous television is an example of a system made up of a no. of modules eg. speakers, projection tube, volume button, channel buttons etc. Each has a functionality of their own but when assembled together make a television.

Q.9 Write a short note on:

(i) **Unified modelling language (UML)**

(ii) **Object Oriented Analysis modelling**

[RTU 2015]

Ans.(i) Unified Approach : Refer to Q.6.

Ans.(ii) Object Oriented Analysis modelling : Object-oriented analysis and modeling or Object-oriented analysis and design (OOAD) is a popular technical approach for analyzing, designing an application, system, or business by applying the object-oriented paradigm and visual modelling throughout the development life cycles to foster better stakeholder communication and product quality.

Although it is possible to do object-oriented development using a waterfall model, in practice most object-oriented systems are developed with an iterative approach. As a result in object-oriented processes "analysis and design" are often considered at the same time.

The object-oriented paradigm emphasizes modularity and re-usability. The goal of an object-oriented approach is to satisfy the "open closed principle".

In object-oriented programming, the open/closed principle states "software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification" that is, such an entity can allow its behaviour to be extended without modifying its source code.

A module is open if it supports extension. That is, the module provides standardized ways to add new behaviors or describe new states. In the object-oriented paradigm this is often accomplished by creating a new subclass of an existing class. A module is closed if it has a well defined stable interface that all other modules must use and that limits the interaction and potential errors that can be introduced into one module by changes in another. In the object-oriented paradigm this is accomplished by defining methods that invoke services on objects. Methods can be either public or private, i.e., certain behaviors that are unique to the object are not exposed to other objects. This reduces a source of many common errors in computer programming.

The purpose of any analysis activity in the software life-cycle is to create a model of the system's functional requirements that is independent of implementation constraints.

The main difference between object-oriented analysis and other forms of analysis is that by the object-oriented approach we organize requirements around objects, which integrate both behaviors (processes) and states (data) with. In other or traditional analysis methodologies, the two aspects: processes and data are considered separately.

In the object-oriented analysis phase of software development, the system requirements are determined, the classes are identified and the relationships among classes are identified.

The three analysis techniques that are used in conjunction with each other for object-oriented analysis are object modelling, dynamic modelling, and functional modelling.

Object Modelling

Object modelling develops the static structure of the software system in terms of objects. It identifies the objects, the classes into which the objects can be grouped into and the relationships between the objects. It also identifies the main attributes and operations that characterize each class.

The process of object modelling can be visualized in the following steps:

- Identify objects and group into classes
- Create the relationships among classes
- Create user object model diagram
- Define user object attributes
- Define the operations that should be performed on the classes
- Review glossary

Dynamic Modelling

After the static behavior of the system is analyzed, its behavior with respect to time and external changes needs to be examined. This is the purpose of dynamic modelling.

Dynamic Modelling can be defined as "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".

The process of dynamic modelling can be visualized in the following steps:

- Identify states of each object
- Identify events and analyze the applicability of actions
- Construct dynamic model diagram, comprising of state transition diagrams
- Express each state in terms of object attributes
- Validate the state-transition diagrams drawn

Functional Modelling

Functional Modelling is the final component of object-oriented analysis. The functional model shows the processes that are performed within an object and how the

data changes as it moves between methods. It specifies the meaning of the operations of object modelling and the actions of dynamic modelling. The functional model corresponds to the data flow diagram of traditional structured analysis.

The process of functional modelling can be visualized in the following steps:

- Identify all the inputs and outputs
- Construct data flow diagrams showing functional dependencies
- State the purpose of each function
- Identify constraints
- Specify optimization criteria

Q.10 Explain the following in context of UML

- Use case diagrams.
- Sequence diagram
- Classes and objects.
- Interfaces.
- State diagrams.

[RTU 2014]

Ans. (i) Use Case Diagrams : Refer to Q.6.

(ii) Sequence Diagram : Refer to Q.3.

(iii) Classes and Objects : Refer to Q.4.

(iv) Interfaces : Interface is an UML representation that is denoted by a circle and name of interface is written below the circle.

Interface describes functionality without implementation. Implementation of an interface is done by the class through the function according to the requirement.



Fig.

(v) State Diagrams : Refer to Q.5.

Q.11 Explain object oriented (OO) concept with an example. [RTU 2013]

Ans. After the analysis phase, the conceptual model is developed further into an object-oriented model using object-oriented design (OOD). In OOD, the technology-independent concepts in the analysis model are mapped onto implementing classes, constraints are identified, and interfaces are designed, resulting in a model for the solution domain. In a nutshell, a detailed description is constructed specifying how the system is to be built on concrete technologies. The stages for object-oriented design can be identified as:

- Definition of the context of the system

Software Engineering

- Designing system architecture
 - Identification of the objects in the system
 - Construction of design models
 - Specification of object interfaces
- ORD introduces a new set of terminology, notation, and procedures for the derivation of a software design.

Objects, Operations and Messages

To accomplish object-oriented design, we must establish a mechanism for (1) the representation of data structure (2) the specification of process, and (3) the invocation procedure.

An example, typical objects might be machines, commands, files, displays, switches, signal, alphanumeric strings, or any other person, place, thing, occurrence, role, or event. When an object is mapped into its software realization, it consists of a private data structure and processes, called operations, that may legitimately transform the data structure. Operations contain control and procedural constructs that may be invoked by message a request to the object to perform of its operations.

The object also has a shared part that is its interface. Messages move across the interface and specify what operation on the object is desired, but not how the operation is to be performed. The object that receives a message determines how the requested operation is to be implemented.

Design Issues

There are five criteria for judging a design method's ability to achieve modularity and related these to object-oriented design.

- (1) Decomposability : The facility with which a design method helps the designer to decompose a large problem into subproblems that are easier to solve.
- (2) Composability : The degree to which a design method ensures that program components (modules), once designed and built, can be reused to create other systems.
- (3) Understandability : The ease with which a program component can be understood without reference to other information or other modules.
- (4) Continuity : The ability to make small changes in a program and have these changes manifest themselves with corresponding changes in just one or a very few modules.
- (5) Protection : An architectural characteristic that will reduce the propagation of side effects if an error does occur in a given module.

From these criteria, five basic design principles can be derived for modular architectures: (1) linguistic modular units; (2) few interfaces (3) small interfaces (weak coupling) (4) explicit interfaces (5) information hiding.

Classes, Instances and Inheritance

All objects are members of a larger class and inherit the private data structure and operations that have been defined for that class. States another way, a class is a set of objects

that each has the same characteristics. An individual object is therefore an instance of a larger class.

The use of classes, subclasses, and inheritance is crucially important in modern software engineering. Reuse of program components (our ability to achieve composition) is attained by creating objects (instances) that build on existing attributes and operations inherited from a class or subclass, we only need to specify how the new object differs from the class, rather than defining all the characteristics of the new object.

Object Descriptions

A design description of an object (an instance of a class or subclass) can take one of two forms.

- (1) A protocol description that establishes the interface of an object by defining each message that the object can receive and that related operation that the object performs, when it receives the message.
- (2) An implementation description that shows implementation details for each operation implied by a message that is passed an object. Implementation details includes information about the object's private part, that is, internal details about the data structure and procedural details that describe operations.

The protocol description is nothing more than a set of

messages and a corresponding comment for each message. An implementation description of an object provides the internal ("hidden") details that are required for implementation but are not necessary for invocation.

An implementation description is comprised of the following information: (1) a specification of the object's name and reference to a class (2) a specification of private data structure with an indication of data items and types (3) a procedural description of each operation or, alternatively pointers to such procedural descriptions.

Q.12 Describe the unified approach to OO design. [RTU 2013]

Ans. The Unified Approach

A unified approach evolved from the combined efforts of Booch, Rumbaugh and Jacobson. This unified approach is known as UML.

UML is organized into two major design activities System design and Object design.

Modeling is simplification of reality. It is a central part of all the activities that leads to the deployment of good software. We build models to communicate the desired structure and behavior of the system. In short, building models help in:

- Visualization and control of system's architecture.
- Exposure of opportunities for simplification and reuse.

- Managing risk.
- Communication for the desired structure and behavior of the system.

So, modeling is very important to be done. In the concern of complexity of a program, the modeling plays an important role to reduce it. As we know that when the levels of abstraction increases, there are more chances of failure and it will be very difficult to solve the problem.

The Unified Modeling Language (UML) is a standard language for writing software blueprints. UML may be used to visualize, specify, construct and document the artifacts of software-intensive system.

Conceptual Model Of UML

UML requires three major elements to build its conceptual model. It includes:

- UML's basic building blocks,
- Rules that dictate how those building blocks may be put together, and
- Some common mechanisms that apply throughout UML.

One more term required to build UML is view. A view is a subset of UML modeling construct, that represents an aspect of the system. One or two kinds of diagrams provide a visual notation for the concepts in each view.

UML can be used for domains like:

- Enterprise Information Systems
- Banking and Financial Services
- Telecommunications
- Transportation Management
- Defence/Aerospace and Navigation
- Scientific and Engineering Applications

1.3 Explain the difference between structural and OO analysis with the help of suitable example. [RTU 2013]

1.3. The Systems Development Life Cycle (SDLC) can be defined as a process of understanding how an information system can support business needs or requirements of an organization, modeling the business processes, designing the systems components, and building the system. A systems development project thus goes through a sequence of some fundamental phases such as planning, analysis, design, and implementation. Each of these phases can be divided into a series of steps or activities that rely on some models and techniques to produce required deliverables. Even though all phases cycle through some common phases or activities, how they are approached by the systems development team can be different - the project team might move through phases or steps logically, consecutively, incrementally, or iteratively.

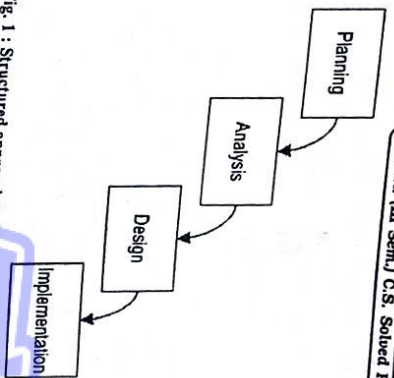


Fig. 1 : Structured approach to SDLC (Waterfall Model)
The most common approach to SDLC is a structured methodology based on Waterfall model. It adopts a formal step-by-step approach to the SDLC phases and activities - the activities of one phase must be completed before moving to the next phase. See Figure 1. At the completion of each activity or phase, a document is produced that must be approved by the stakeholders before moving to the next activity or phase. The structured approach looks at a system from a top-down view. The center of the structured approach is the process model, which depicts the business processes of a system, and the primary model that presents the processes is the data-flow diagram (DFD). The DFDs and their associated data dictionary contain information about the systems components (inputs, outputs, processes, and data storage) that need to be designed and ultimately built.

Object-Oriented Approach

Another approach to SDLC that is widely discussed in recent years is the object-oriented methodology. It is developed by the software engineering professionals who deal with large and complex systems in domains such as aerospace and process control. Failure of a business system may cost money but that of mission-critical systems such as that used in the airplanes and space shuttles might cost life. Business system is only one domain that is typically addressed in the SAD texts.

The object-oriented methodology views a system as a bottom-up approach to systems development. To start with, it describes the system through a set of business processes it performs as well as the object classes that these processes deal with. It uses a set of diagrams or models to represent various views and functionality of a system and is commonly known as Unified Modeling Language or UML. When these models are used along with a particular method of systems development, the OO approach later became known as the Unified process.

Software Engineering

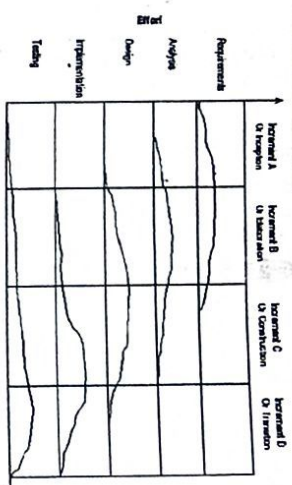


Fig. 2: Iterative and Incremental Model of SDLC

Unified Process follows an iterative and incremental approach to systems development. The systems development life cycle is viewed as consisting of several increments or phases: inception, elaboration, construction, and transition. In each increment or phase, the developers move through the activities of gathering requirements, analyzing the requirements, designing the system, implementing the design, and testing the system. Thus the Unified approach is a two-dimensional model. See Figure 2. As shown, the phases of the traditional systems development approach do not match with those of the Unified Process; but in each increment, all phases of the SDLC (requirements, analysis, design, implementation, and testing) are visited until the developers satisfy the requirements. However, in each increment, activities of one phase predominate over the others-causing the systems development effort to move from the inception to elaboration, from elaboration to construction, and from construction to transition.

PART-C

Q.14(a) Write similarities and dissimilarities between object oriented and functional oriented design approaches.

(b) Perform Object-Oriented Analysis and design for the following problem:

A factory has machines that fail uniformly after continuous operation and needs frequent adjustments and repair after a certain Mean Time To Failure (MTTF). A manager has certain number of adjusters who keep the machines running. The manager maintains a queue of operative machines and idle adjusters. If machines are waiting to be repaired, then the manager assigns the first queued machine to the next available adjuster. The factory

administrator wants to get maximum possible output from its machines and adjusters. The objective of the simulation is to see how average machine and adjuster utilization depends on:

- Number of Machines
- Number of Adjusters
- Reliability of Machines in terms of MTTF
- Productivity of Adjusters

[R.T.U. 2018]

Ans(a) Difference Functional-Oriented Design versus Object-Oriented Design

No.	Functional-Oriented Design (FOD)	Object-Oriented Design (OOD)
1.	The basic abstractions, which are given to the user, are real world functions.	The basic abstractions are not the real world functions but are the data abstraction where the real world entities are represented.
2.	Functions are grouped together by which a higher level function is Page on obtained using of this technique is SASD.	Functions are grouped together on the basis of the data they operate since the classes are associated with their methods.
3.	In this approach the state information is often represented in a centralized shared memory.	In this approach the state information is not represented in a centralized memory but is implemented or distributed among the objects of the system.
4.	FOD approach is mainly used for computation sensitive application.	Whereas OOD approach is mainly used for evolving system which mimics a business process or business case.
5.	We decompose in function/procedure level.	We decompose in class level.
6.	Top down approach.	Bottom up approach.
7.	It views system as black box that performs high level function and later decompose it detailed function so to be mapped to modules.	Object-oriented design is the discipline of defining the objects and their interactions to solve a problem that was identified and documented during object-oriented analysis.
8.	Begins by considering the use case diagrams and scenarios.	Begins by identifying objects and classes.

ans.(b) "Fail uniformly" probably means that they fail at a constant rate (with a different rate for each type of machine). If a machine fails uniformly in this context as it implies the machine failure time follows a uniform distribution (this would roughly the same number of machines failing in every time period, which is not realistic).

Failing at a constant rate per unit time happens when number of fails X in the interval from 0 to t (so λt fails on average) follows a poisson distribution

$$P(X = k) = (\lambda t)^k \exp(-\lambda t) / k!$$

If roughly half of these machines have failed up to some time t_0 then half of the remaining will fail in the next interval up to t_0 and so on, decreasing all the time. If T is the time to failure of a machine, which is a random variable, then the probability that the machine is still working after time t is the same as the probability that X is 0 i.e. there has been no fail up to time t . Then

$$P(T > t) = P(X = 0) = \exp(-\lambda t)$$

This is the reliability of the machine type, which is the proportion of machines which are still working after time t . The failure rate λ is $1/\text{MTTF}$ for that type of machine. So for given MTTF we can work out the proportion of machines which will fail up to time t using the reliability. e.g. If lightbulbs have a MTTF of 300hrs then the proportion which are still working after 200hrs is

$$\exp(-1/300 \times 200) = 0.513 \text{ or } 51\%$$

The proportion which have failed up to time t is $1 - \text{reliability}$. This can be written as an integral up to time t of the probability density for T . Differentiation will then give the density for the variable T , but the reliability is a more meaningful thing to focus on for practical purposes.

5 Write short notes on :

(a) Object Oriented Analysis modelling

(b) ODD concepts and methods

[Note : ODD read as 'OOD']

[R.T.U. 2017]

OR
Explain Object oriented design concepts and methods.

[RTU 2015, 2014]

(a) Refer to Q.9(ii).

ans.(b) Refer to Q.11.

6 Explain Object Oriented Analysis and its Approach and Explain class and Object Relationship Model.

OR

[R.T.U. 2016]

Discuss Object Oriented Analysis (OOA) and modelling in detail.

OR

[RTU 2014]

Describe the Class-Responsibility-Collaborator (CRC).

[RTU 2013]

B.Tech. (III Sem.) C.S. Solved Papers

OR

Describe Object Oriented Analysis Process. What is CRC modelling? Describe in detail.

[RTU 2009, 2008; Raj. Univ. 2005, 2002]

OR

Explain with diagram the four steps of object oriented analysis modelling.

Ans. Object Oriented Analysis Process : In the OOA process, first the scenario of the usage of the system has to be understood then modelling of software begins. These are described as follows:

Use Cases : Use cases were introduced into object analysis by Jacobson as a means of communication with users. They are scripts that describe typical ways that a system is or will be used. A use case in general is the sense that it doesn't describe instances of how a system is used, but generalizes a number of instances into a general script. Use cases should achieve the following objectives :

- (1) To define the functional and operational requirements of the system (product) by defining a scenario of usage that is agreed upon by the end-user and the software engineering team.
 - (2) To provide a clear and unambiguous description of how the end-user and the system interact with one another.
 - (3) To provide a basis for validation testing.
- During OOA, use cases serve as the basis for the first element of the analysis model. In developing a use case, we identify the main actors or roles in the system and then describe the way they use or interact with the system or their behaviour in the system. Use case identification usually begins by looking at scenarios to check how these actors carry out their activities. We then use these scenarios to identify the usual way in which these actors carry out their activities. A system if usually described by a number of use cases, which together become the use case model.

Class-Responsibility-Collaborator Modelling

After developing usage scenarios, the system classes, their responsibilities and collaborators are identified. Class-responsibility-collaborator provides a simple way to identification and arrangement (organization) of classes that are relevant to system or product requirements. Ambler has described CRC Modelling as :

"A CRC model is really a collection of standard index cards that represent classes. The cards are divided into three sections. Along the top of the card we write the name of the class. In the body of the card we list the class responsibilities on the left and the collaborators on the right"

Software Engineering

In reality, the CRC model may make use of actual or virtual cards, so that an organized representation of classes can be developed. Responsibilities are the attributes and operations that are relevant for the class. Collaborators are those classes that are needed to provide a responsibility.

Classes : As we know that the object can be external entities, things, occurrences or events, roles; organizational units, places or structures. These are identified in the context of a software problem. All nouns become potential objects.

Coad and Yourdon suggested five selection characteristics that should be used as analyst considers each potential object for inclusion in the analysis model. The various selection characteristics are :

1. Retained Information : Potential objects will be useful during analysis only if information about it must be remembered so that the system can function.
2. Needed Services : Potential object must have a set of identifiable operations that can change the value of its attributes in some way.
3. Common Attributes : A set of attributes can be defined for the potential object and these attributes apply to all occurrences of the object.
4. Common Operations : A set of operations can be defined for the potential object and these operations apply to all occurrences of the object.
5. Essential Requirements : External entities that appear in the problem space and produce or consume the information is essential to the operation of any solution for the system will almost always be defined as objects in the requirements model.

To be considered a legitimate object for inclusion in the requirements model, a potential object should satisfy some (or almost all) of these characteristics. Firesmith has extended this class types by suggesting some additions, as given below :

- Device class model external entities.
 - Property classes represent some important property of the problem environment.
 - Interaction classes model interactions that occur among other objects.
- In addition, objects and classes may be categorized by a set of characteristics that are Tangible (tangible or intangible class), Inclusive (atomic or aggregate class), Sequence (sequential or concurrent class), Persistent (temporary, permanent or transient class) and Integrity (Secured or unsecured class). Using these class categories, the index card might be extended to include the type of class and its characteristics as shown.

Class Name

Class Type : (like device, property)

Class Characteristics (like tangible, inclusive)

Collaborations

Responsibilities : Responsibilities basically consist of attributes and operations of a class. The attributes are the stable features of a class and can be achieved from the statement of the scope or by understanding the nature of the class. The operations of class exhibit the behaviour of the class. Wirfs-Brock and her colleagues suggested some

1. System intelligence should be evenly distributed.
2. Each responsibility should be stated as general as possible.
3. Information and the behaviour related to it should reside within the same class.
4. Information about one thing should be localized within a single class, not distributed across multiple classes.
5. Responsibilities should be shared among related classes, when appropriate.

The responsibilities of classes can be fulfilled in two ways

1. A class can use its own operations to manipulate its own attributes.
 2. A class can collaborate with other class.
- Collaboration : Wirfs-Brock and her colleagues define collaborations in the following way :

"Collaborations represent requests from a client to a server in fulfillment of a client responsibility. A collaboration is the embodiment of the contract between the client and the server. We say that an object collaborates with another object to fulfill a responsibility. It needs to send the other object any messages. A single collaboration flows in one direction - representing a request from the client to the server. From the client's point of view, each of its collaborations are associated with a particular responsibility implemented by the server.

The collaborations are used to identify relationship between classes. When a set of classes all collaborate to achieve some requirement, they can be organized into a sub-system. Collaborations are identified by determining whether a class can fulfill each responsibility itself. It can't, then it needs to interact with another class.

To aid in the identification of collaborators, the analyst can examine three different generic relationships between classes, given by Wirfs-Brock, these are explained below one-by-one.

1. The collaboration is part of relationship in which one class is the part of another class. For eg, the class human body is part of a class human.

2. The knowledge of relationship is required when one class acquires information from another class.

3. It depends upon relationship exists in which two classes have a dependency with each other. For eg, human head must always be connected to human body.

So, human head depends upon human body. So, the analyst develops the connections necessary to identify these relationships. In all cases, the index card contains a list of responsibilities and the corresponding collaborations that enable the responsibilities to be fulfilled. After developing a complete CRC model, the model is reviewed by using the following approach given by Ambler :

(1) All participants in the review (of the CRC model) are given a subset of the CRC model index cards. Cards that

collaborate should be separated (*i.e.* no reviewer should have two cards that collaborate).

(2) All use case scenarios (and corresponding use case diagrams) should be organized into categories.

(3) The review leader reads the use-case deliberately. As the review leader comes to a named object, she passes a token to the person holding the corresponding class index card.

(4) When the token is passed, the holder of the class card is asked to describe the responsibilities noted on the card. The group determines whether one (or more) of the responsibilities satisfies the use-case requirement.

(5) If the responsibilities and collaborations noted on the index, (Cards can't accommodate the use case) are made to the cards.

When all use cases (or use case diagrams) have been reviewed, OOA continues.

□□□