**Q1. Compare and contrast the Booch and Rumbaugh methodologies.**

Ans.

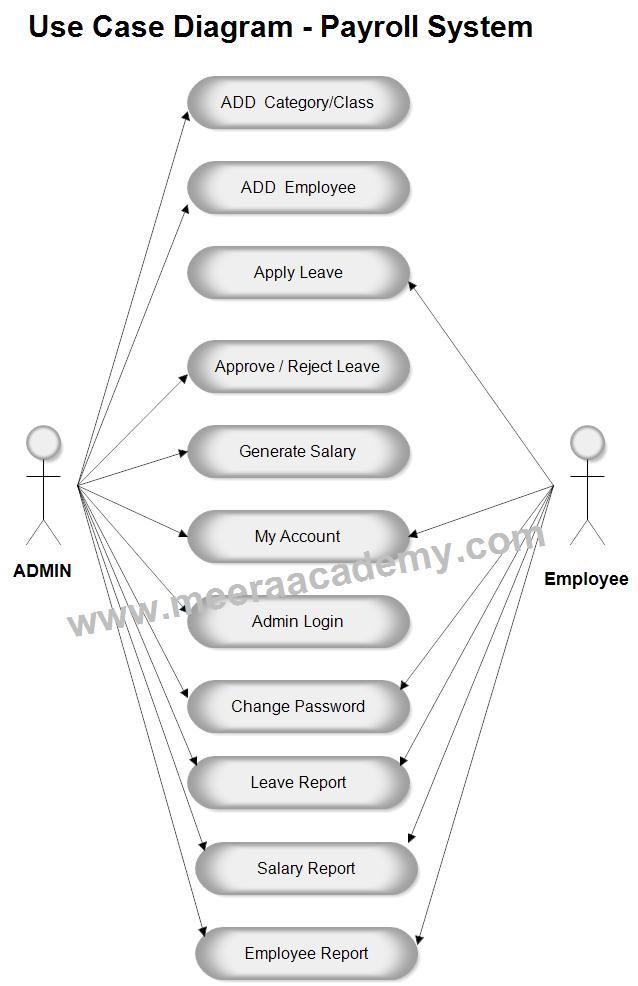
|  |  |  |  |
| --- | --- | --- | --- |
|  | ***Booch Method*** | ***Rumbaugh Method*** |  |
| ***Approach:***    ***Phases Covered:***    ***Strength:***      ***Weakness:***    ***Uni-directional Relationship:***  ***Bi-directional Relationship:***  ***Diagrams used:*** | Object centered approach.  Analysis, design and implementation phases.  Strong method for producing detailed object oriented design models.    Focus entirely on design and not on analysis.  Uses.    Associations.    Class diagram, state transition diagram, object diagram, timing diagram, Module diagram, process diagram. | Object centered approach.  Analysis, design and implementation phases.  Strong method for producing object model static structure of the system.    Cannot fully express the requirements.    Directed Association.    Uni-directed Associations.  Data flow diagrams, state transmission diagram, class/object diagram. |  |

**Q2. Explain Use-case driven approach in object-oriented system development with the payroll system as a case study.**

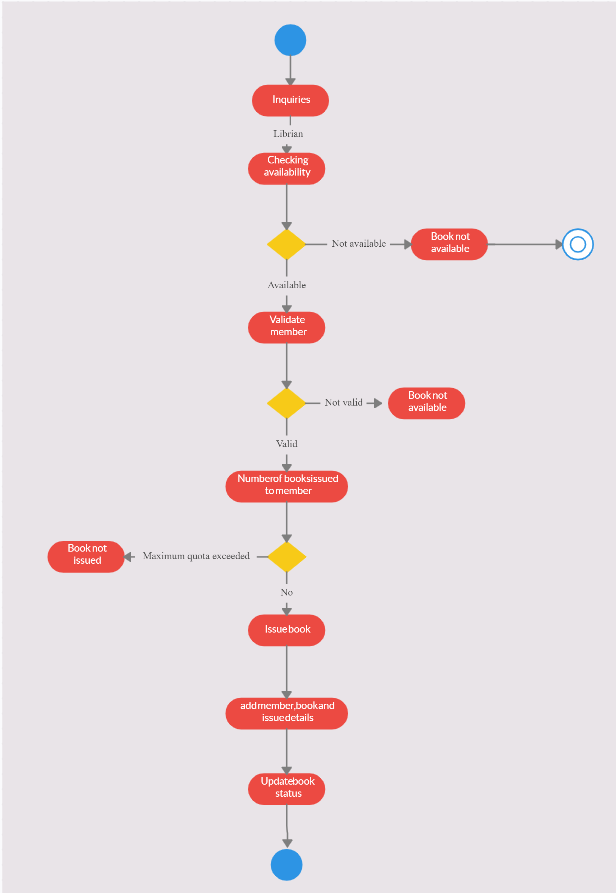
Ans. A **use case** is a sequence of actions, performed by one or more **actors** (people or non-human entities outside of the system) and by the system itself, that produces one or more results of value to one or more of the actors. One of the key aspects of the Unified Process is its use of use cases as a driving force for development. The phrase *use case driven* refers to the fact that the project team uses the use cases to drive all development work, from initial gathering and negotiation of requirements through code. (See "Requirements" later in this chapter for more on this subject.)

Use cases are highly suitable for capturing requirements and for driving analysis, design, and implementation for several reasons.

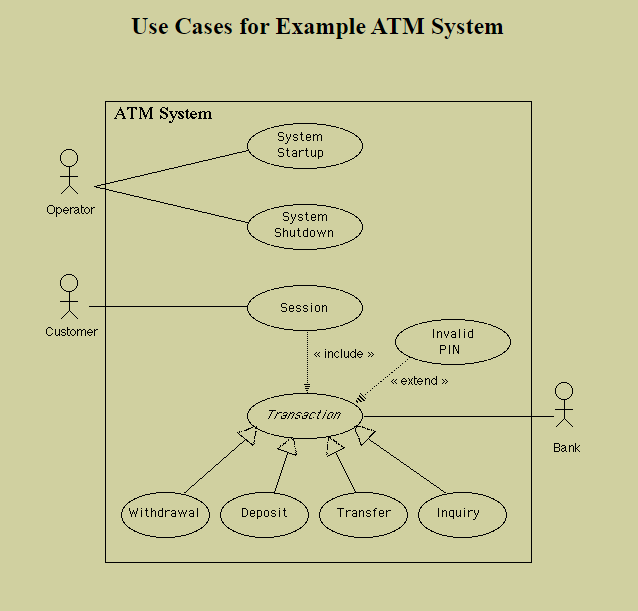
* Use cases are expressed from the perspective of the system's users, which translates into a higher comfort level for customers, as they can see themselves reflected in the use case text. It's relatively difficult for a customer to see himself or herself in the context of requirements text.
* Use cases are expressed in natural language (English or the native language of the customers). Well-written use cases are also intuitively obvious to the reader.
* Use cases offer a considerably greater ability for everyone to understand the real requirements on the system than typical requirements documents, which tend to contain a lot of ambiguous, redundant, and contradictory text. Ideally, the stakeholders should regard use cases as binding contracts between customers and developers, with all parties agreeing on the system that will be built.
* Use cases offer the ability to achieve a high degree of traceability of requirements into the models that result from ongoing development. By keeping the use cases close by at all times, the development team is always in touch with the customers' requirements.



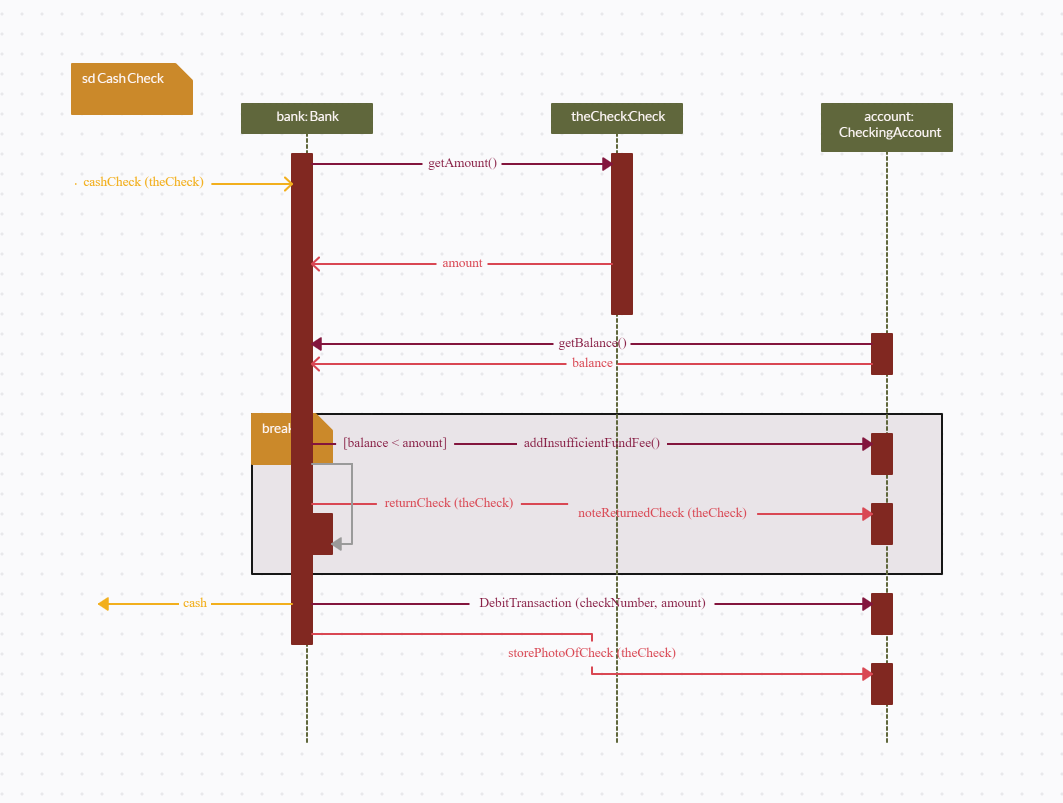
**Q3. Develop an activity diagram to show the business process of the library.**

Ans. 

**Q4. Show the activities involved in an ATM transaction.**

Ans. 

**Q5. Develop sequence / collaboration diagrams for the following use case of the banking system: Identify all the classes, responsibilities and collaborators processes for the objects. (i) Deposit in saving account (ii) Withdrawal from saving account (both acceptance and denial) (iii) Balance checking.**

Ans. 

**Q6. What is object-oriented SDLC? Compare it with traditional approaches.**

Ans. The Object-Oriented approach of Building Systems takes the objects as the basis. For this, first the system to be developed is observed and analyzed and the requirements are defined as in any other method of system development. Once this is often done, the objects in the required system are identified.

Object-oriented model employs an object-oriented strategy. The primary objectives are:

**1.** Object-oriented analysis,

**2.** Object-oriented design,

**3.** Object-oriented programming

Object-oriented analysis develops an object-oriented model of the application domain. Object-oriented design develops an object-oriented model of the software system. Object oriented programming realizes the software design with an object-oriented programming language that supports direct implementation of objects, classes, and inheritance.

There are a variety of object-oriented methodologies such as:

* **Object Identification:**  
  System objects and their characteristics and events.
* **Object Organization:**  
  Shows how objects are related via “part-of” relationships.
* **Object Interfaces:**  
  Shows how objects interact with other objects.

**Advantages of Object-Oriented Life Cycle Model:**

* Design is no longer carried out independently of the later implementation because during the design phase we must consider which components are available for the solution of the problem.
* Design and implementation become more closely associated.
* Duration of the implementation phase is reduced.
* A new job title emerges, the class librarian, who is responsible for ensuring the efficient usability of the class library.

**Q7. Describe in detail the major and minor elements of object model. Give suitable examples.**

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

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

* Abstraction
* Encapsulation
* Modularity
* Hierarchy

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

* Typing
* Concurrency
* Persistence

Abstraction

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

Grady Booch has defined abstraction as follows −

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

**Example** − When a class Student is designed, the attributes enrolment\_number, name, course, and address are included while characteristics like pulse\_rate and size\_of\_shoe are eliminated, since they are irrelevant in the perspective of the educational institution.

Encapsulation

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

Modularity

Modularity is the process of decomposing a problem (program) into a set of modules so as to reduce the overall complexity of the problem. Booch has defined modularity as −

“Modularity is the property of a system that has been decomposed into a set of cohesive and loosely coupled modules.”

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

Hierarchy

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

The two types of hierarchies in OOA are −

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

Typing

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

The two types of typing are −

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

Concurrency

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

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

Persistence

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

**Q8. How is the use case model useful in every phase of software development? Discuss.**

##### Ans. **1. Use Cases as a better form of documentation**

Instead of one monolithic Software Requirement Specification (SRS) in a single document, using Use Cases encourages us to split up our SRS into more focused smaller documents which together cover all of the requirements a traditional SRS would contain. Each Use Case has a Use Case Specification document which focuses exclusively on the functional requirements for that particular Use Case. If we need to change a particular piece of functionality, we only need to change the Use Case that functionality is expressed in, and won’t need to go through the documentation for the entire project!

As a consequence, change control and configuration management are made easier. Instead of having to distribute one massive document to everyone on the project every time a minor change is made, we can now version control each individual Use Case Specification and the Supplementary Specification.

##### **2. Use Cases as a tool for communication**

Since a good Use Case Specification is one that is written from the user’s perspective, it serves as a useful document for discussing the functional requirements of the system being developed with everyone involved on the project. It should be something that the end-user (customer) is able to read and understand relatively easily, while still giving the analysts, designers, developers and tester’s sufficient understanding for doing their jobs. A Use Case Specification is not intended as a highly technical document, but rather as a document that all parties involved on the project can use to agree the scope and functionality of the project.

##### **3. Use Cases as a tool for Project Management**

Having your Software Requirement Specification captured in the form of Use Case Specifications and a Supplementary Specification, there are several additional benefits for Project Management. Since we are defining the functional requirements in terms of Use Cases, we are effectively breaking down the system into smaller parts in terms of functional decomposition. While this does not mean that all functionality will be exclusive to a particular Use Case (some functionality, or rather some implementations of functionality, might be shared across Use Cases), it certain lends itself to parallel development for disparate pieces of functionality.

The first phase of any project involves identifying all of the Use Cases for the project, and doing some initial analysis on them with the aim to identify those Use Cases that pose the greatest risk to the projects overall chance of success or failure. The idea would then be to tackle the high-risk Use Cases in early iterations in the project to eliminate project risk. If the risk cannot be eliminated early on then the project will still fail, but with as little amount of wasted effort and expenditure as possible. When used in conjunction with an iterative process, Use Cases give us an excellent way of managing project risk.

##### **4. Use Case Driven Development**

Using UML is not a requirement for using Use Cases to capture functional requirements for a system. However, you will often find that project teams that employ Use Cases also use UML to model the system being developed. There are established techniques for using the flow of events in a Use Case Specification to define analysis versions of Sequence and Communication Diagrams, which in turn are used in identifying analysis classes, the results of which directly feed into the design process wherein design classes are identified and fleshed out. Those design classes are then used by the programmers to implement the system. The whole procedure forms part of what is known as “**Use Case Driven Development**”, where Use Cases are used to drive the development process.

**Q9. Draw the use case model for the following system and explain. Also use noun phrase approach to identify the classes and their attributes. Explain the process.**

**A computerized banking network includes both human cashiers and automatic teller machines (ATM) to be shared by a consortium of banks. Each bank provides its own computer to maintain its own accounts and process transactions against them. Cashier stations are owned by individual banks and communicate directly with the bank computers. Human cashiers enter account and transaction data ATM accepts a cash card, interacts with the user, communicates with the central computer to carry out the transaction, dispenses the cash and prints receipts. The system requires appropriate record keeping and security provisions. The system must handle concurrent accesses to the same account correctly.**

Ans. We now read through the requirements document and underline the nouns (or noun phrases) and circle (italicize) the verb phrases.

Design the **software** *to support* a computerized **banking network** including both **human** **cashiers** and **automatic teller machines** (ATMs) *to be shared by* a **consortium** of banks.  Each **bank** *provides* its own **computer** *to maintain* its own **accounts** and *process* **transactions** against them.  **Cashier stations** *are owned by* individual **banks** and *communicate* *directly with* their own **bank’s computers**.  **Human cashiers** *enter* account and transaction **data**.  **Automatic teller machines** *communicate with* a **central computer** which *clears* **transactions** with the appropriate banks.  An **automatic teller** **machine** *accepts* a **cash card**, *interacts with* the **user**, *communicates with* the **central** **system** *to carry out* the **transaction**, *dispenses* **cash**, and *prints* **receipts**.  The **system** *requires* appropriate **record-keeping and security provisions**.  The **system** *must handle* **concurrent accesses** to the same account correctly.  The **banks** *will provide* their own **software** for their own **computers**; you are to design the **software** for the ATMs and the **network**.  The **cost** of the shared system *will be apportioned* to the **banks** according to the **number of customers** with **cash cards**.

**List of underlined nouns:**

software           banking network           cashier              ATM                consortium

bank                 bank computer             account            transaction        cashier station

account data     transaction data            central              cash card          user

                                                            computer

cash                 receipt                          system              record-keeping

                                                                                    provision

Security            access                          cost                  customer

provision

**From this list of nouns we need to identify the classes.  We eliminate candidates that are:**

1.      Redundant --    user    (customer)

2.      Irrelevant --      cost

3.      Implementation constructs --     words like algorithm, process, data structure, interrupt, linked list, and CPU cycle are not part of the problem domain, but may be relevant in the design and implementation of the system.  In this list we may exclude --    access,             software

4.      Roles – the name of a class should reflect its intrinsic nature and not a role it plays in an association.  In a **company**a **person** may be asked to play the role of a **boss** or a **subordinant**.  A system could be modeled that included these two separate classes, but a system in which a person object was capable of playing either role would be far more robust and flexible.  In this list there are no terms that should be considered as roles in an association.

5.      Attributes – names that describe properties of individual objects and names that do not have additional attributes and behaviors associated with them. --           account data,                transaction data

      (Both of these are not simple strings or numbers but have several fields and could be considered objects)

6.      Operations – the name of something that is applied to an object, but not manipulated in its own right.  None in this list.

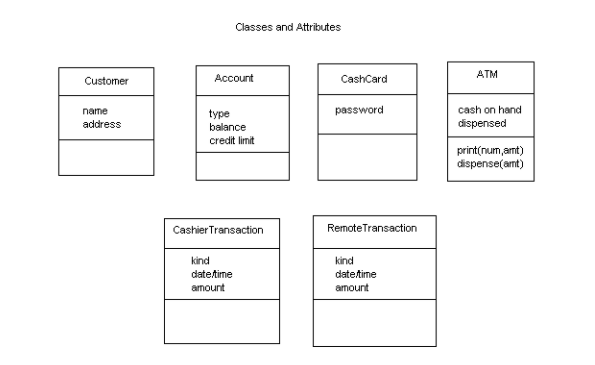
7.      Vague classes – classes must be specific with well-defined properties and boundaries.  Names describing general concepts are not good candidates for being classes.  Vague nouns may indicate areas in the requirements document that need further refinement, or point to subsystems that will need to be fleshed out in subsequent refinements of the model.  In this example the vague nouns are:                     system,      security provision,   record-keeping provision,               banking network

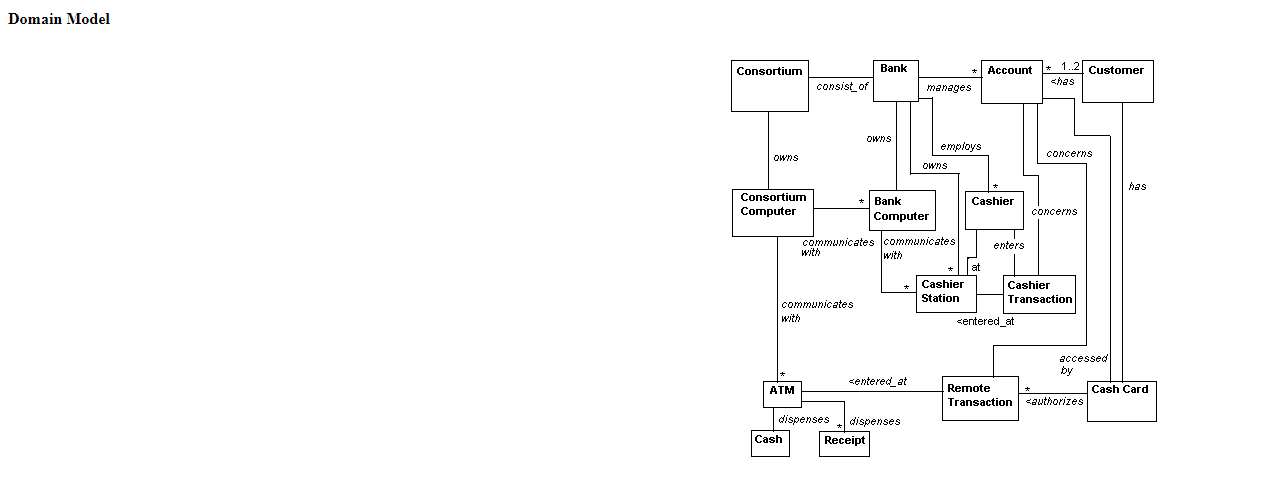
**Identified classes in the ATM system**

Account           ATM                Bank                Bank-computer            CashCard

Cashier Consortium       Customer         CashierStation              CentralComputer

Transaction       Cash                Receipt





**Q10. Describe the components of the unified approach with a neat diagram.**

Ans. The unified approach to software development revolves around (but is not limited to) to the following processes and concepts. The processes are:

1.     Use-case driven development

2.     Object-oriented analysis

3.     Object-oriented design

4.     Incremental development and prototyping

5.     Continuous testing

The UA allows iterative development by allowing you to go back and forth between the design and the modelling or analysis phases. It makes backtracking very easy and departs from the linear waterfall process, which allows no form of back tracking.

## i.      Object-Oriented Analysis

Analysis is the process of extracting the needs of a system and what the system must do to satisfy the users’ requirements. The goal of object-oriented analysis is to first understand the domain of the problem and the system’s responsibilities by understanding how the users use or will use the system. It concentrates on describing what the system does rather than how it does it. Separating the behavior of a system from the way it is implemented require viewing the system from the user’s perspective rather than that of the machine. OOA process consists of the following steps:

1.     Identify the Actors.

2.     Develop a simple business process model using UML Activity diagram.

3.     Develop the Use Case.

4.     Develop interaction diagrams.

5.     Identify classes.

**ii.      Object-Oriented Design**

Booch, provides the most comprehensive object-oriented design method. Ironically, since it is so comprehensive, the method can be somewhat imposing to learn and especially tricky to figure out where to start. Rumbaugh et al.‘s and Jacobson et al.’s high-level models provide good avenues for getting started. UA combines these by utilizing Jacobson et al.’s analysis and interaction diagrams,Booch’s object diagrams, and Rumbaugh et al.’s domain models. Furthermore, by following Jacobson et al.’s life cycle model, we can produce designs that are traceable across requirements, analysis, design, coding, and testing. OOD Process consists of:

1.     Designing classes, their attributes, methods, associations, structures and protocols, apply design axioms.

2.     Design the Access Layer

3.     Design and prototype User interface

4.     User Satisfaction and Usability Tests based on the Usage/Use Cases

5.     Iterated and refine the design

#### iii.      Iterative Development and Continuous Testing

You must iterate and reiterate until, eventually, you are satisfied with the system. Sine testing often uncovers design weaknesses or at least provides additional information you will want to use, repeat the entire process, taking what you have learned and reworking your design or moving on the prototyping and retesting. Continue this refining cycle through the development process until you are satisfied with the results. During this iterative process, your prototypes will be incrementally transformed into the actual application. The UA encourages the integration of testing plans from day 1 of the project. Usage scenarios can become test scenarios; therefore, use case will drive the usability testing. Usability testing is the process in which the functionality of software is measured.

**Q11. What are the models proposed by OMT? Give their purpose.**

Ans. **Object Modelling Technique (OMT)** is real world based Modelling approach for software Modelling and designing. It was developed basically as a method to develop object-oriented systems and to support object-oriented programming. It describes the static structure of the system.

Object Modelling Technique is easy to draw and use. It is used in many applications like telecommunication, transportation, compilers etc. It is also used in many real world problems. OMT is one of the most popular object-oriented development techniques used now-a-days. OMT was developed by *James Rambaugh*.

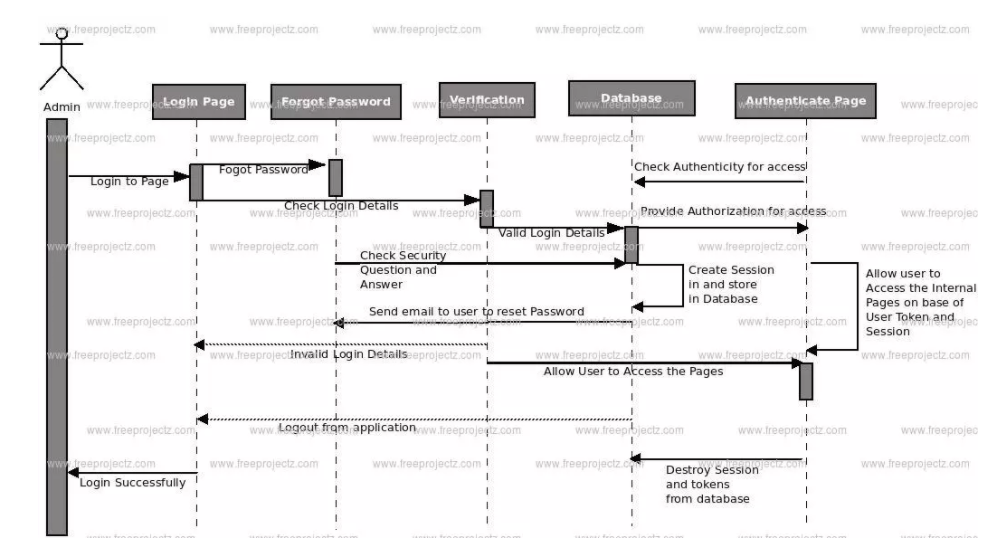
**Purpose of Object Modelling Technique:**

* To test physical entity before construction of them.
* To make communication easier with the customers.
* To present information in an alternative way i.e. visualization.
* To reduce the complexity of software.
* To solve the real world problems.

**Object Modelling Technique’s Models:**  
There are three main types of models that has been proposed by OMT:

1. **Object Model:**  
   Object Model encompasses the principles of abstraction, encapsulation, modularity, hierarchy, typing, concurrency and persistence. Object Model basically emphasizes on the *object* and *class*. Main concepts related with Object Model are classes and their association with attributes. Predefined relationships in object model are aggregation and generalization (multiple inheritance).
2. **Dynamic Model:**  
   Dynamic Model involves states, events and state diagram (transition diagram) on the model. Main concepts related with Dynamic Model are states, transition between states and events to trigger the transitions. Predefined relationships in object model are aggregation (concurrency) and generalization.
3. **Functional Model:**  
   Functional Model focuses on the how data is flowing, where data is stored and different processes. Main concepts involved in Functional Model are data, data flow, data store, process and actors. Functional Model in OMT describes the whole processes and actions with the help of data flow diagram (DFD).

**Q12. Consider “buy tickets” use case in a railway reservation system. Draw a sequence diagram. Explain briefly.**

Ans. 

**Q13. Explain in detail the various properties of object-oriented systems. Illustrate with suitable examples.**

Ans. An object-oriented system revolves around a Class and objects. A class is used to describe characteristics of any entity of the real world. An object is a pattern of the class. An actual object created at runtime is called as an instance. A class, apart from characteristics has some functions to perform called as methods. For.e.g A class named “Food” has attributes like ‘price’, ‘quantity’. “Food” class has methods like Serve\_food(), bill\_food().  
  
**Object:** Objects in Object Oriented Systems interact through messages.  
  
**Inheritance:** The main class or the root class is called as a Base Class. Any class which is expected to have ALL properties of the base class along with its own is called as a Derived class. The process of deriving such a class is Derived class. For the “Food” class, a Derived class can be “Class Chinesefood”.  
  
**Abstraction:** Abstraction is creating models or classes of some broad concept. Abstraction can be achieved through Inheritance or even Composition.  
  
**Encapsulation:** Encapsulation is a collection of functions of a class and object. The “Food” class is an encapsulated form. It is achieved by specifying which class can use which members (private, public, protected) of an object.  
  
**Polymorphism:** Polymorphism means existing in different forms. Inheritance is an example of Polymorphism. A base class exists in different forms as derived classes. Operator overloading is an example of Polymorphism in which an operator can be applied in different situations.

**Q14. The Small Library Database System will be used by the Biology Department of a local college to track the borrowing of books and other forms of media, such as video tapes, and software. A secretary will operate the system and will responsible for checking out books to students and faculty members. Identify objects and relationships among them using Noun phrase approach. Describe the approach used.**

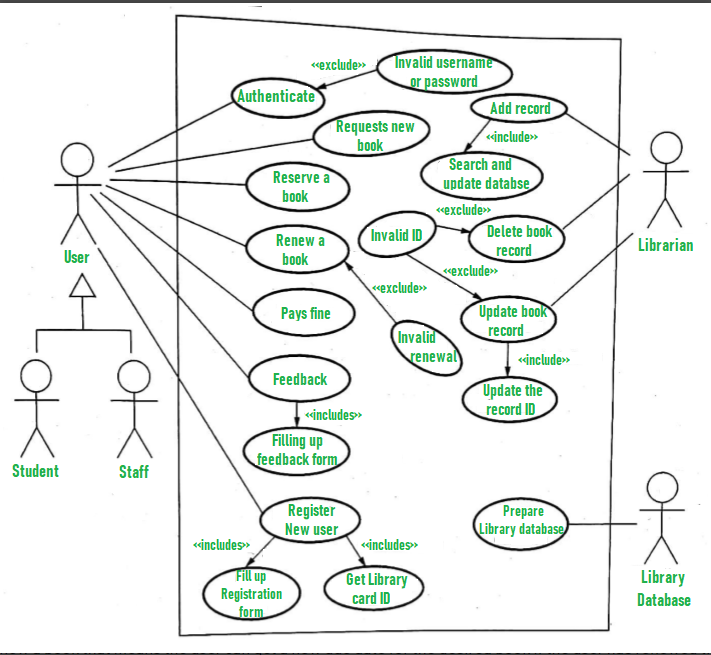
Ans. (VARIABLE)

**Q15. What is the use of use-case diagram? Differentiate between the roles of actors and users.**

Ans. The purpose of use case diagram is to capture the dynamic aspect of a system. Additional diagrams and documentation can be used to provide a complete functional and technical view of the system. They provide the simplified and graphical representation of what the system must actually do.

The difference between an actor and an individual system user is that an actor represents a particular class of user rather than an actual user. Several users can play the same role, which means they can be one and the same actor. In that case, each user constitutes an instance of the actor.

**Q16. Draw and explain the use-case diagram for a library management system.**

Ans. 

1. User who registers himself as a new user initially is regarded as staff or student for the library system.
   * For the user to get registered as a new user, registration forms are available that is needed to be fulfilled by the user.
   * After registration, a library card is issued to the user by the librarian. On the library card, an ID is assigned to cardholder or user.
2. After getting the library card, a new book is requested by the user as per there requirement.
3. After, requesting, the desired book or the requested book is reserved by the user that means no other user can request for that book.
4. Now, the user can renew a book that means the user can get a new due date for the desired book if the user has renewed them.
5. If the user somehow forgets to return the book before the due date, then the user pays fine. Or if the user forgets to renew the book till the due date, then the book will be overdue and the user pays fine.
6. User can fill the feedback form available if they want to.
7. Librarian has a key role in this system. Librarian adds the records in the library database about each student or user every time issuing the book or returning the book, or paying fine.
8. Librarian also deletes the record of a particular student if the student leaves the college or passed out from the college. If the book no longer exists in the library, then the record of the particular book is also deleted.
9. Updating database is the important role of Librarian.

**Q17. Briefly explain the Object-Oriented system Development Life Cycle.**

Ans. (SAME AS Q6.)

**Q18. Discuss briefly the Booth Methodology for object modelling.**

Ans. In software engineering the Booch method,that is published in 1991 by **Grady Booch**, is a widely used method in object-oriented analysis and design.

The Booch method has been superseded by UML, which features elements from the Booch method with OMT and OOSE.

The Booch method helps to design systems using the object paradigm. It covers the analysis- and design phases of an object-oriented system. The method defines different models to describe a system and it supports the iterative and incremental development of systems.

The Booch method includes **six types of diagrams** such as class diagrams, object diagrams, state transition diagrams, module diagrams, process diagrams and interaction diagrams.

# The Booch method notation



|  |
| --- |
|  |
|  |
| Figure 1. A class diagram notation. |



|  |
| --- |
|  |
|  |
| Figure 2. A object diagram notation. |

The dynamic nature of an application can be illustrated by state transition and interaction diagrams.



|  |
| --- |
|  |
|  |
| Figure 3. An interaction diagram. |

There are several Booch diagrams that are very similar to diagrams in UML. These Booch diagrams are state transition and interaction diagrams. The State transition diagram corresponds to UML's statechart diagram and the interaction diagram corresponds to UML's sequence diagram.

**Q19. Briefly explain about Class diagram with an example.**

Ans. The [**UML**](https://en.wikipedia.org/wiki/Unified_Modeling_Language) Class diagram is a graphical notation used to construct and visualize object oriented systems. A class diagram in the Unified Modelling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's:

* classes,
* their attributes,
* operations (or methods),
* and the relationships among objects.
* A class diagram shows a set of classes, interfaces, and collaborations and their relationships.
* most common diagram found in modeling object-oriented systems.
* address the **static design** view of a system.
* Class diagrams that include active classes address the static process view of a system.

**Q20. Explain different types of relationships among classes and objects.**

* Ans. There are four kinds of relationships in the UML:
  + 1. Dependency *(a semantic relationship between two things in which a change to one thing may affect the semantics of the other thing)*

2. Association *(a structural relationship that describes a set of links, a link being a connection among objects)*

3. Generalization *(a specialization / generalization relationship in which objects of the specialized element (the child) are substitutable for objects of the generalized element (the parent))*

4. Realization (a semantic relationship between classifiers, wherein one classifier specifies a contract that another classifier guarantees to carry out) (between interfaces and the classes or components and between use cases and the collaborations)

**Q21. Describe the activities of object-oriented system development life cycle.**

Ans. (SAME AS Q6. & Q17.)

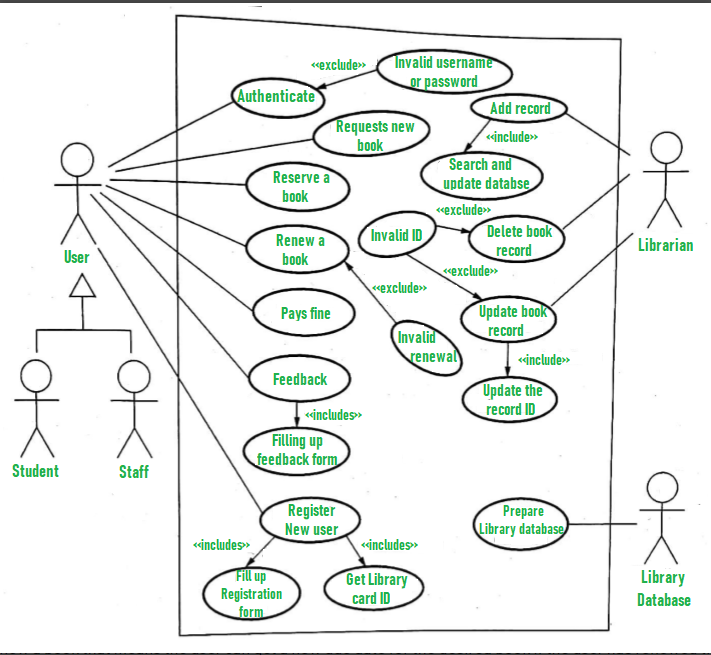
**Q22. Explain Rumbaugh’s methodology of object-oriented development process.**

Ans. The Rumbaugh method is listed first because it is these authors favourite, and we find it a very friendly and easy methodology.  For traditional system analyst's, the Rumbaugh's methodology is the closest to the traditional approach to system analysis and design, and beginners will recognize familiar symbols and techniques.  The Rumbaugh methodology has its primary strength in object analysis but it also does an excellent job with object design. Rumbaugh has three deliverables to the object analysis phase; the Object model, the Dynamic model, and the functional model.  These three models are similar to traditional system analysis, with the additions for the object model, including definitions of classes along with the classes variables and behaviours. The Rumbaugh object model is very much like an entity relationship diagram except that there are now behaviours in the diagram and class hierarchies.  The dynamic model is a "state transition" diagram that shows how an entity changes from one state to another state. The functional model is the equivalent of the familiar data flow diagrams from a traditional systems analysis.

**Q23. Identify the use-cases, classes, and relationships for the Bank ATM system. Also give the use case diagram and class diagram for the Bank ATM system.**

Ans. (SIMILAR TO Q9.)

**Q24. Explain the noun phrase approach to identify the classes for a library information system.**

Ans. 

Use the Noun Phrase Approach

**Q25. What are the structural and behavioural UML diagrams?**

Ans. **Structural Diagrams –**

**Composite Structure Diagram:** It shows the internal structure of a classifier, classifier interactions with the environment through ports, or behaviour of a collaboration.

**Deployment Diagram:** It shows a set of nodes and their relationships that illustrates the static deployment view of an architecture.

**Package Diagram:** It groups related UML elements into a collection of logically related UML structure.

**Profile Diagram: Profile diagram** is [**structure diagram**](https://www.uml-diagrams.org/uml-25-diagrams.html#structure-diagram) which describes **lightweight extension mechanism** to the UML by defining custom [**stereotypes**](https://www.uml-diagrams.org/stereotype.html), [**tagged values**](https://www.uml-diagrams.org/stereotype.html#tagged-value), and constraints.

**Class Diagram:** It shows a set of classes, interfaces, and collaborations and their relationships, typically, found in modelling object-oriented systems.

**Object Diagram:** It shows a set of objects and their relationships, which is the static snapshots of instances of the things found in class diagrams.

**Component Diagram:** It shows a set of components and their relationships that illustrates the static implementation view of a system.

**Behavioural Diagrams-**

**Activity Diagram:** It is a graphical representation of workflows of stepwise activities and actions with support for choice, iteration and concurrency

**Use Case Diagram:** It describes a system’s functional requirements in terms of use cases that enable you to relate what you need from a system to how the system delivers on those needs.

**State Machine Diagram:** It shows the discrete behaviour of a part of a designed system through finite state transitions.

**Sequence Diagram:** It shows the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario.

**Communication Diagram:** It shows interactions between objects and/or parts (represented as lifelines) using sequenced messages in a free-form arrangement.

**Interaction Overview Diagram:** It depicts a control flow with nodes that can contain other [**interaction diagrams**](https://en.wikipedia.org/wiki/Interaction_diagram).

**Timing Diagram:** It shows interactions when the primary purpose of the diagram is to reason about time by focusing on conditions changing within and among lifelines along a linear time axis.

**Q28. What are the approached used for identification of classes and attributes? Explain.**

Ans.  **The four alternative approaches for identifying classes:**

* The noun phrase approach.
* The common class patterns approach.
* The use-case driven, sequence/collaboration modeling approach.
* The classes, responsibilities and collaborators (CRC) approach.

**Noun phrase approach**

* Look for the noun phrases through the use cases.
* Three categories:
* Relevant classes.
* Fuzzy classes.
* Irrelevant classes.
* Identifying tentative classes.
* Look for noun phrases and nouns in the use cases.
* Some classes are implicit or taken from general knowledge.
* All classes must make sense in the application domain.
* Carefully choose and define class names.
* Selecting classes from the relevant and fuzzy classes.
* Redundant classes.
* Adjective classes.
* Attribute classes.
* Irrelevant classes.

**The common class patterns approach.**  
The common class patterns approach is based on a knowledge base of the common classes that have been proposed researchers. The patterns used for  
finding the candidate class and object are:

* Concept class
* Events class
* Organization class
* People class
* Places class
* Tangible things and devices class

**The classes, responsibilities and collaborators (CRC) approach.**  
Classes, responsibilities, and collaborators is a technique used for identifying classes’ responsibilities, and collaborators and therefore their attributes and methods. Furthermore, CRC can help us identify classes. CRC is based on the idea that an object either can accomplish a certain responsibility itself or it may require the assistance of other objects.  
CRC cards are 4”``X 6`` index cards in which all the information for an object is written is cheap, portable, readily available and familiar.

|  |  |
| --- | --- |
| Class Name Responsibilities … | Collaborators … |

**Steps in CRC process**  
The classes, responsibilities and collaborators process consists of three steps:

* Identify classes’ responsibilities (and identify classes).
* Assign responsibilities.
* Identify collaborators.

