Management Group (OMG) and has been managed by this organization ever since. In 2005, UML was also published by the International Organization for Standardization (ISO) as an approved ISO standard. Since then the standard has been periodically revised to cover the latest revision of UML.

7.2.1 UML Diagrams

Mostly, the UML diagrams are used for developing different types of requirements models and system models. The UML has a set of different types of diagrams but the most commonly used UML diagrams for modeling are as follows:

i) **Activity Diagram:** This diagram shows the flow of activities described by the use case (or activities that are involved in a process or data processing). An activity diagram is similar to a DFD (data-flow diagram).

ii) **Swimlane Diagram:** This diagram is a variation of the activity diagram. It is used to represent the flow of activities described by the use case and at the same time indicate which actor or analysis class has responsibility for the action described by an activity rectangle.

iii) **Use Case Diagram:** This diagram shows the interactions between a system and its environment, i.e. external actors (users or other systems/devices). A use case can be taken as a simple scenario that describes what a user expects from a system. Each use case represents a separate task that involves external interaction with a system.

iv) **Sequence Diagram:** This diagram also shows interactions between the actors and the objects in a system and the interactions between the objects themselves. The interactions are shown in sequential order, i.e. the order in which these interactions take place. Sequence diagrams represent how operations are carried out in a system.

v) **Collaboration Diagram:** This diagram is also an interaction diagram. It is also called a *communication diagram*. In the analysis model (requirements model) or system model, a collaboration diagram is used to show how classes collaborate with one another.

vi) **Class Diagram:** This diagram shows the object classes in the system and the associations between these classes. An association is a link between classes that indicates a relationship between these classes. The purpose of the class diagram is to represent the static view of the system.

vii) **State Diagram:** This diagram shows how the system reacts to internal and external events. The state diagram also shows system states and events that cause transitions from one state to another.

ANALYSIS MODEL

An analysis model is a technical representation of the system or software to be built. It s that the analysis model represents the information, functions, and behavior of the software

system to be built. Afterward, the analysis model is translated into the architecture, interface, and component-level design in the design modeling.

In analysis modeling, system requirements are represented in a way that is relatively easy to understand and straight forward to review for correctness, completeness, and consistency. The analyst or software engineer builds the model using requirements collected (elicited) from the customer. The analysis model acts as a bridge between system description and the design model. Figure 7.1 shows the role of the analysis model between system description and software design.

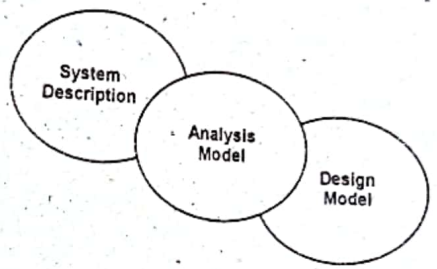


Figure 7.1: Analysis Model as a bridge between System Description & Design Model

Objectives of Analysis Model

The analysis model must achieve three primary objectives:

i) to describe the requirements of the customer,

ii) to establish a way of creating software design (i.e. design model), and

iii) to define a set of requirements that can be validated, once the software is built.

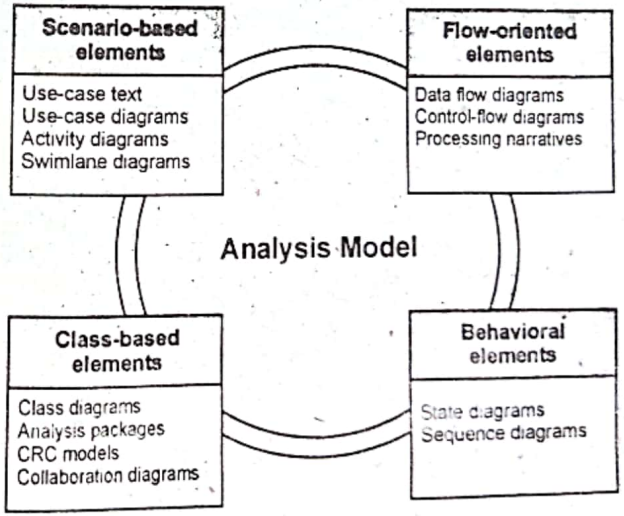


Figure 7.2: Elements of the Analysis Model

7.3.1 Elements of Analysis Model

There are many different ways to look at the requirements for a computer-based system. Different modes of representation can be used to consider requirements from different viewpoints. The generic elements that are common to most analysis models are shown in figure 7.2.

7.3.1.1 Scenario-Based Elements

The system is described from the user's point of view using a scenario-based approach. Scenario-based elements of the requirements model are often the first part of the model that is developed. So, they serve as input for the creation of other modeling elements. Scenario-based elements consist of use cases in text form (which have been discussed in chapter 6), use-case diagrams, activity diagrams, and swimlane diagrams.

1. Use-Case Diagram

A use case represents a sequence of activities that involves actors (i.e. users or other sub-systems/devices) and the system. UML use case diagrams are mostly used to represent the interaction between a system and external actors (users or other devices). Figure 7.3 shows a simple use case as a part of the library system.

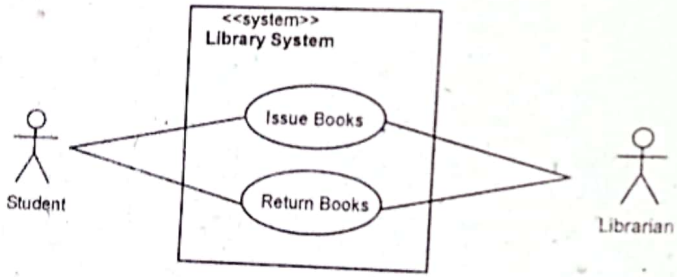


Figure 7.3: UML use case diagram in a library system

In a UML use case diagram, the commonly used notations are as follows:

- A stick figure notation () is used to represent an actor. An actor can be an internal or external entity (a human or any other system or object) that interacts with the system.
- A *use case* is represented as an ellipse shape "" with a name inside it.

2. Activity Diagram

UML activity diagram shows the flow of activities described by the use case (or activities that are involved in a process or data processing). An activity diagram is similar to a DFD (data-flow diagram). Figure 7.4 shows the UML activity diagram for reserving a ticket.

3. Swimlane Diagram

UML swimlane diagram is a variation of the UML activity diagram. It is used to represent the flow of activities described by the use case and at the same time indicate which actor or analysis class has responsibility for the action described by an activity rectangle. Responsibilities are represented as parallel segments that divide the diagram vertically, like the lanes in a swimming pool. Figure 7.5 shows the swimlane diagram for reserving a ticket.

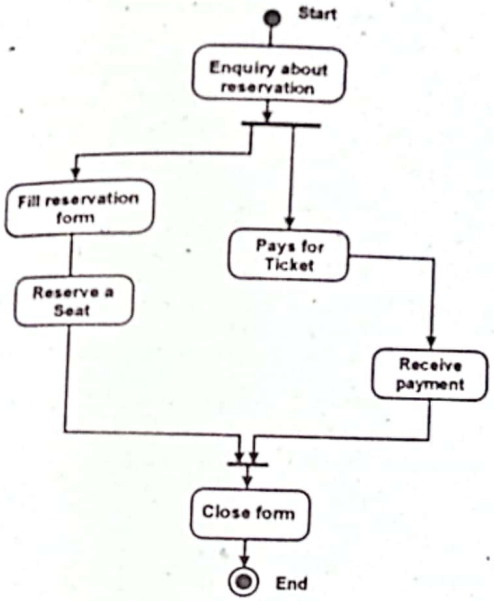


Figure 7.4: UML activity diagram

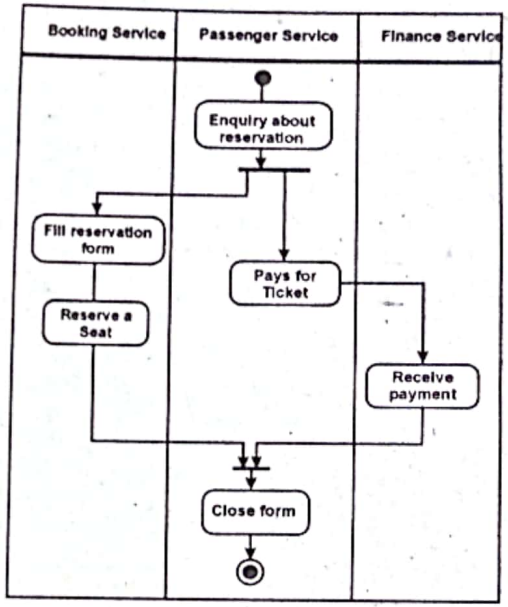


Figure 7.5: UML Swimlane diagram

The notations used for the UML activity diagram/swimlane diagram are as follows:

- The start of a diagram is indicated by a filled circle such as .
- The end of a diagram is indicated by a filled circle inside another circle such as  .
- Rectangles with round corners represent activities that are carried out.
- Arrows represent the flow of control from one activity to another. An arrow may be annotated with a proper message to indicate the condition when that flow is taken.
- A diamond symbol is used between two activities with two outgoing paths. The flow of control moves to one of two paths based on the given condition. If the condition is true, the flow of control moves toward that path; otherwise, the second path.

Note: *The activity diagram and use-case diagram are discussed in detail in chapter 8.*

7.3.1.2 Class-Based Elements

A class is an object-oriented concept. Each usage scenario suggests a set of objects that are manipulated as an actor interacts with the system. These objects are categorized into classes. A class is a collection of things that have similar attributes and common behaviors. Class-based elements consist of class diagrams, analysis packages, CRC models, and collaboration diagrams.

1. Class Diagrams

UML class diagrams are used when developing an object-oriented system model to show the classes in a system and the associations/relationships between these classes. Class diagrams are the main building blocks of object-oriented modeling. An object class can be thought a general definition of one kind of system object. A class diagram provides a static or structural view of a system. It does not show the dynamic nature of the communications between the objects of the classes in the diagram (The class diagram is discussed in detail in chapter 8 with examples).

The main elements of a class diagram are boxes or rectangles, which are the icons used to represent classes and interfaces. Each box is divided into horizontal parts or sections:

- The top section contains the name of the class.
- The middle section contains a list of attribute names of the object of the class.
- The third section contains the operations or behaviors of the class. An operation refers to what objects of the class can do. It is usually implemented as a method of the class (In Java, C++, and other object-oriented programming languages, operations are called *methods*).

The general structure of a class is shown in figure 7.6.

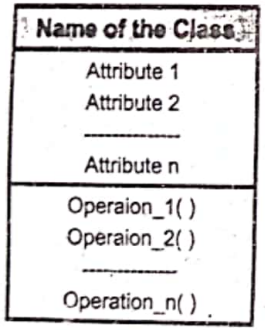


Figure 7.6: Structure of a class

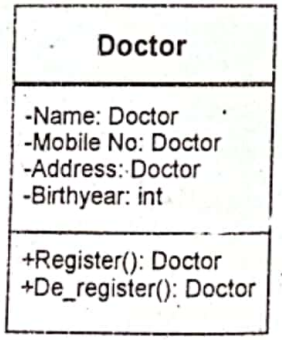


Figure 7.7: Doctor class

Figure 7.7 shows a simple example of a Doctor class. It has four attributes (Name, Mobile, Address, and Birthyear) and two operations (Register() and De_register()). Each attribute can have a name, a type, and a level of visibility. The type and visibility are optional. The type follows the name and is separated from the name by a colon. The visibility is indicated by a preceding –, #, ~, or +, indicating, private, protected, package,

or public visibility, respectively. In figure 7.7, all attributes have private visibility, as indicated by the leading minus sign (–). Each operation can also have a level of visibility, parameters with names and types, and a return type.

Note: The class diagrams are discussed in detail in chapter 8.

2. Analysis Package

Categorization is an important part of analysis modeling. It means that various elements of the analysis model (e.g., use cases, analysis classes) are categorized in a manner that packages them as a grouping. It is given a representative name. This mechanism is called an *analysis package*. For example, Classes such as Tree, Road, Bridge, and Building are given the representative name "Environment". Similarly, Classes such as Department, Faculty Member, and Student are given the representative name "University". These classes can be grouped into analysis packages as shown in figure 7.8.

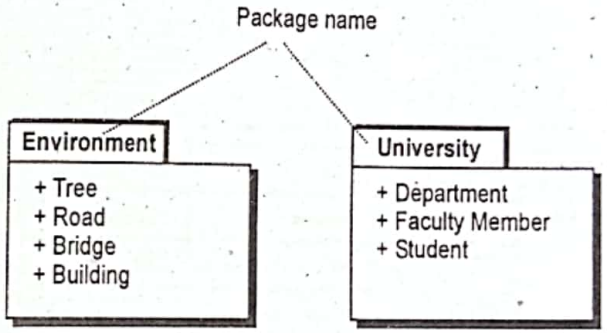


Figure 7.8: Packages

The plus sign preceding the analysis class name in each package indicates that the classes have public visibility and are, therefore, accessible from other packages. The minus (-) sign and # symbol can also be used preceding the analysis class name. A minus indicates that an element is hidden from all other packages and a # symbol indicates that an element is accessible only to packages contained within a given package.

3. CRC Models

CRC stands for Class Responsibility Collaborator. CRC modeling provides a simple means for identifying and organizing the classes that are relevant to the system or product requirements. Ambler describes CRC modeling in the following way:

"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, the name of the class is written. In the body of the card, class responsibilities are written on the left and the collaborators are written on the right." The CRC card layout is shown in figure 7.9.

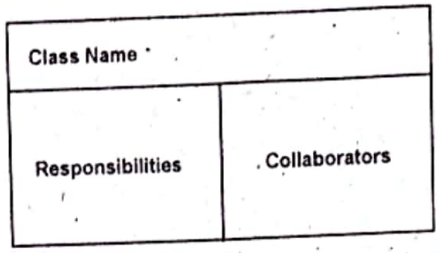


Figure 7.9: CRC card layout

The CRC model may make use of actual or virtual index cards. The aim is to develop an organized representation of classes. Responsibilities are the attributes and operations that are relevant to the class. A responsibility is "anything the class knows or does". Collaborators are those classes that are required to provide a class with the information needed to complete a responsibility. Figure 7.10 shows a simple CRC index card for the class "ATM CardReader".

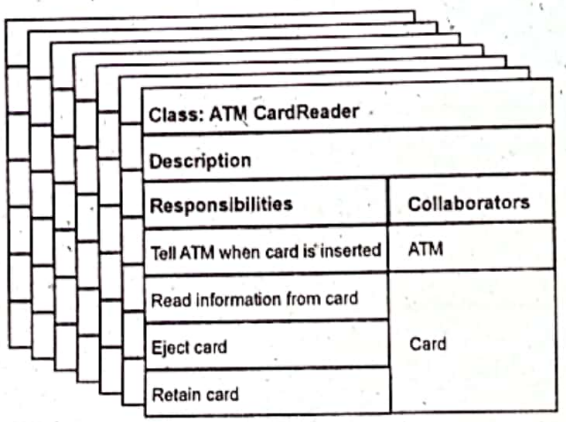


Figure 7.10: The CRC model index card

4. Collaboration Diagrams

UML collaboration diagram is an interaction diagram. It is also called a *communication diagram*. In the analysis model (requirements model), a collaboration diagram is used to show how analysis classes collaborate with one another. In other words, a collaboration diagram is used to show the relationships between objects and the order of messages passed between objects within a system. As component-level design proceeds, it is sometimes useful to show the details of these collaborations by specifying the structure of messages that are passed between objects within a system.

Figure 7.11 illustrates a simple collaboration diagram for logging into the student management system. This collaboration diagram consists of three objects Login,

Authentication, and Database. Messages are passed between objects as illustrated by the arrows in figure 7.11. Each message includes a unique sequence number. The flow of communication in the above collaboration diagram is as follows:

1. A student requests a login through the login object.
2. An authentication object checks the request.
3. If a student entry exists in the database, then the access is allowed; otherwise, an error is returned.

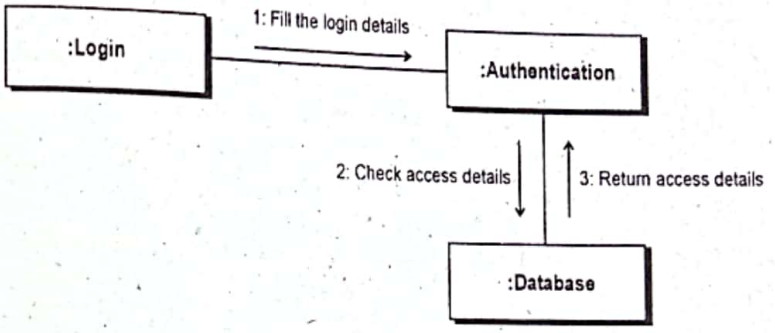


Figure 7.11: Collaboration diagram for logging into the system

7.3.1.3 Behavioral Elements

The behavior of a computer-based system can have a deep effect on the design that is chosen and the implementation approach that is applied. Therefore, the requirements model must provide modeling elements that show the behavior of the system. Behavioral elements represent the states of the system and how they are changed by internal and external events. Behavioral elements consist of state diagrams and sequence diagrams.

1. State Diagrams

UML state diagram is used to represent the behavior of a system. It shows how the system reacts to internal and external events. The state diagram also shows system states and events that cause transitions from one state to another.

2. Sequence Diagrams

Sequence diagrams are used to represent interactions between the actors and the objects in a system and the interactions among the objects themselves. The interactions are shown in sequential order, i.e. the order in which these interactions take place.

Note: *UML State and Sequence diagrams are discussed with examples in detail in chapter 8.*

7.3.1.4 Flow-Oriented Elements

Information flows through a computer-based system it gets transformed. It shows how the data objects are transformed while they flow between the various system functions. The flow elements are data flow diagrams (DFDs), control flow diagrams (CFDs), and processing narratives.

1. Data Flow Diagrams

A data flow diagram (DFD) is a graphical representation of the flow of data (or processing steps) through a system or process. UML does not support a data-flow diagram (DFD). The UML activity diagram is similar to the data-flow diagram (DFD). So, the UML activity diagram can be used as an alternative to DFD. The detail of DFD is given in the next topic.

2. Control Flow Diagrams

A control-flow diagram (CFD) is a diagram to describe the control flow of a business process or system. This diagram helps to understand the detail of a process. It shows where control starts and ends and where it may branch off in another direction, etc. Control-flow diagrams were introduced in the 1950s, and are widely used in multiple engineering disciplines.

3. Processing Narratives

A narrative is a way of presenting connected events in order to tell a good story. In software engineering, a processing narrative is similar to the use case in style but somewhat different in purpose. The processing narrative provides an overall description of the function to be developed. It is not a scenario written from one actor's point of view.

7.3.2 Analysis Rules of Thumb

The rules of thumb have been suggested for analysis modeling that should be followed while creating the analysis model. These rules are as follows:

1. The model focuses on the requirements that are visible within the problem or business domain. The level of abstraction should be relatively high, i.e. there is no need to give details.
2. Each element of the analysis model should add to an overall understanding of software requirements and provide insight into the information domain, function, and behavior of the system.
3. Delay consideration of infrastructure and other non-functional models until design. For example, the database may be initially required for a system, but the classes, functions, and behavior of the database are not initially required. If these are initially considered then there is a delay in the system design.

4. Minimize coupling throughout the system. Coupling is the interconnections between the modules (software components). Coupling is discussed in detail in chapter 10.
5. The requirements model should provide value to all stakeholders. For example,
   - business stakeholders should use the model to validate requirements,
   - designers should use the model as a basis for system design, and
   - software testers should use the model to test the software system.
6. The model should be simple as possible. A simple model always helps to understand the system requirements easily.

7.3.3 Analysis Modeling Approaches

In analysis modeling, analysts use various approaches/methods to describe the software systems. The most popular analysis modeling approaches are structured analysis and object-oriented analysis.

1) Structured Analysis

The structured analysis is a systematic development approach that enables the analyst to understand the overall functionality of the software system and its activities. It considers data and the processes (that manipulate or transform the data) as separate entities. Data objects are modeled in a way that defines the attributes of data objects and relationships among them. Processes that manipulate or transform the data are modeled in a manner that shows how data objects are manipulated/transformed and flow through the system.

During structured analysis, various tools and techniques are used for system development. The most important are as follows:

- Data Flow Diagrams
- Data Dictionary
- Decision Trees
- Decision Tables
- Program Design Language

2) Object-Oriented Analysis (OOA)

The object-oriented analysis focuses on the definition of classes and the manner in which they collaborate with one another. OOA is the first technical activity performed as part of object-oriented software engineering. The main purpose of OOA is to identify the objects of a system that have to be implemented. OOA models are developed during the requirement analysis. The OOA does not include detail of the individual objects in a system to be built.

The following main tools are used in object-oriented analysis and design techniques:

- Class diagrams/templates
- Object diagrams
- Object state diagrams

7.4 DATA MODELING

Data modeling is the process of creating a simplified diagram of a software system and the data elements it contains, to represent the data and how it flows within a system. Data models are created as part of overall requirements modeling.

In data modeling, an entity-relationship diagram (ERD) is used for creating data models. ERD is a graphical tool which is used to represent different data objects and their relationships within a system. A software engineer or analyst defines all data objects and their relationships with the help of ERD. The ERD represents all data objects that are entered, stored, transformed, and produced within an application.

The data model consists of three inter-related pieces of information: data objects, data attributes, and relationships.

7.4.1 Data Objects

The data object is a representation of something (or composite information) that has a number of different properties or attributes. The data object can be an external entity (e.g., anything that produces or consumes information), an event, a place, a role, an organizational unit (e.g., admin department), a structure, etc. For example, a student can be viewed as a data object because he/she has a set of attributes like his name, his height, his address, his color, etc. Anything that has only one attribute is not considered as a valid data object.

St_Code	St_Name	St_Height	St_Color
0303	M. Naseem	6.0"	White
0304	Aqsa	5.2"	White
0305	Hadia	5.1"	White
0306	Iqra	5.3"	White

The headings in the table represent the attributes of the object. The student is defined in terms of St_Code, St_Name, St_Height, and St_Color. This table contains the records of four students. The body of the table consists of instances of the data object. For example, "0303, M. Naseem, 6.0, White" represents an instance of the data object.

7.4.2 Data Attributes

Data attributes define the properties of a data object. They can be used to "name an instance of the data object", "describe the instance", and "make reference to another instance in

another table". One or more of the attributes must be defined as an identifier. The identifier becomes a "key" that can be used to find or access an instance of the data object. In most cases, the value of the identifier is unique, although this is not a requirement. In the above table, St_Code is an example of an identifier.

Attribute Domain

A possible set of values that can be assigned to an attribute of the entity (data object) is called an *attribute domain*. It is also known as a *value set* of an attribute. The attribute domain may consist of a range of values or some specific values. For example, if the "Marks" of a student represents the attribute and its maximum value is 100, we can assign values from 0 to 100 in the Marks attribute. So, the domain for the "Marks" attribute can be from 0 to 100. Similarly, the domain for attribute "Gender" attribute can either be "Male" or "Female". Different attributes can have the same domain.

7.4.3 Relationships

The data objects can be connected to one another in different ways. Normally, they are connected to each other with common identifiers. Consider two objects, "Teacher" and "Student". A connection between these two objects can be created because these two objects are related to each other. Figure 7.12 (a) shows the relationships between data objects "Teacher" and "Student". This relationship is bi-directional. It means that they can be read in either direction. A teacher teaches a student or a student is taught by a teacher. Therefore, a set of object-relationship pairs can be defined that define the relevant relationships. For example,

- A Teacher teaches a Student.
- A Student is taught by a Teacher

The relationships "teaches" and "is taught by" define the relevant connections between Teacher and Student. Figure 7.12 (b) shows this relationship graphically. This figure provides more information about the directionality of the relationship and reduces ambiguity or misinterpretations.

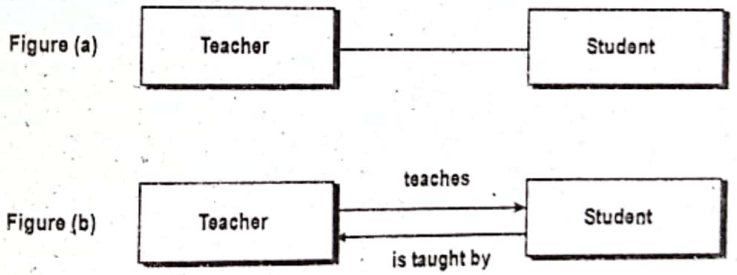


Figure 7.12: Relationships between data objects

7.4.3.1 Cardinality

The data model must be capable of representing the number of occurrences of objects in a given relationship. The *cardinality* of an object-relationship pair specifies the number of occurrences of one object that can be related to the number of occurrences of another object. Two objects can be related as: one-to-one, one-to-many and many-to-many. These relationships are described below.

1. One-to-One (1:1) Relationship

In a one-to-one (1:1) relationship, for each instance of the first object, there is only one matching instance in the second object. On the other hand, for each instance in the second one, there is only one matching instance in the first one. Some examples of one-to-one (1:1) relationships are as follows:

- Consider the relationship between the Country and the Capital. A Country has only one Capital. On the other hand, a Capital belongs to only one Country.
- Now consider the relationship between the Principal and the College. A Principal runs only one College. On the other hand, a College is run by only one Principal.

2. One-to-Many (1:N) Relationship

In one-to-many (1:N) relationship, for each instance in the first object, there can be many matching instances in the second object. On the other hand, for each instance in the second object, there is only one matching instance in the first object. For example, a mother may have many children, but a child can have only one mother.

3. Many-to-Many (M:N) Relationship

In many-to-many (M:N) relationship, for each instance in the first object, there can be many matching instances in the second object. On the other hand, for each instance in the second object, there can also be many matching instances in the first object. For example, an uncle can have many nephews, while a nephew can have many uncles.

7.4.3.2 Modality

In a relationship, the minimum number of instances of one object that may be associated with each instance of another related object is called minimum cardinality. It can either be "0" or "1". The minimum cardinality refers to the modality of a relationship. Modality defines the nature of the relationsl.ip that whether it is optional or mandatory (i.e. compulsory).

- If a relationship has a minimum cardinality "0", then the relationship is called *optional*.
- If a relationship has a minimum cardinality "1", then the relationship is called *mandatory*.

The modality of a relationship is "0" if there is no need for the relationship, i.e. relationship is optional. The modality is "1" if an occurrence of the relationship is required.

7.5 ENTITY-RELATIONSHIP DIAGRAMS (ERD)

The object-relationship pair is the foundation of the data model. These pairs can be represented graphically with the help of an entity-relationship diagram (ERD). It means that ERD represents the data object relationship graphically.

The ERD was introduced by Peter Chen in 1977 for the design of relational database systems. The data objects, attributes, relationships, and various types of symbols are the basic components of the ERD. The primary purpose of the ERD is to represent data objects and their relationships. UML class diagram is similar to ERD. In software engineering, a class diagram is preferred to represent data objects (or object classes) and their relationships/ associations instead of ERD.

In ERD notation, data objects are represented by a labeled rectangle, and the relationship between the objects is represented with lines connecting to the objects. In some ERDs, the connecting line contains a diamond symbol that is labeled with the name of the relationship. The connection between data objects and relationships also uses some special symbols that indicate cardinality and modality. These symbols may be "||" or "O".

Figure 7.13 shows the relationship between the data objects Book and the Author. An Author writes one or many books. On the other hand, a book may be written by one or more Authors. On each side, one instance of the object is necessary. The cardinality of "1" is denoted by a small vertical line (|) on the relationship line, next to the first object or before the second object. This bar indicates that at least one instance is necessary. Similarly, the cardinality of "many" is denoted by the crow's foot (-<) symbol which indicates many instances.

Figure 7.13: ERD and relationship between objects

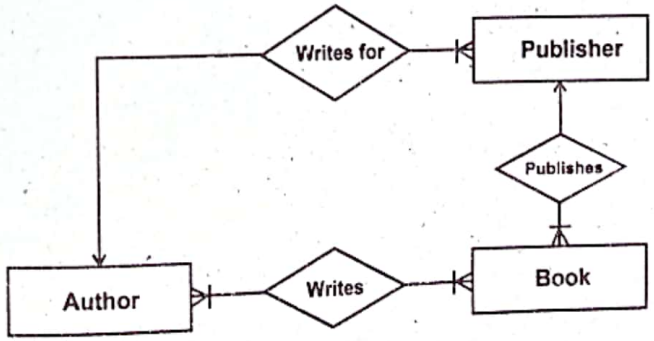


Figure 7.14: An expanded ERD

In the ERD as shown in figure 7.13, the relationship "writes" is indicated by a diamond shape on the connecting line between data objects "Author" and "Book". New data objects can be introduced by expanding the figure 7.13 and these will be related to each other. The data object "Book" may be related to the data object "Publisher". Similarly, the "Author" may be related to "Publisher" and so on. Figure 7.14 shows this relationship. In this figure, the relationships "writes for", "publishes", and "writes" indicate how data objects are associated with each other.

In addition to the basic ERD notations introduced in figure 7.14, the analyst can also represent data objects hierarchically. In many cases, a data object may represent a category of information. For example, the data object "Book" can be categorized as "Computer book", "Math book", "Physics book", etc. Similarly, the object "Computer book" may further be categorized into "Java book", "C++ book", etc. Figure 7.15 shows the ERD notation in the form of a hierarchy.

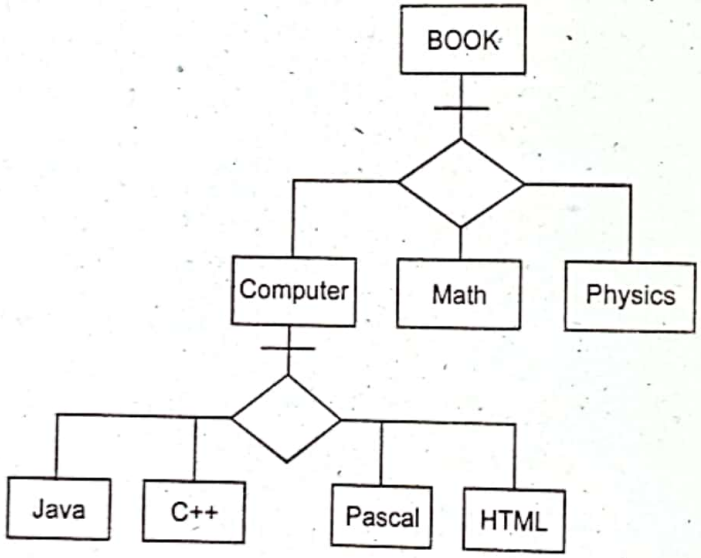


Figure 7.15: Data object in hierarchical form

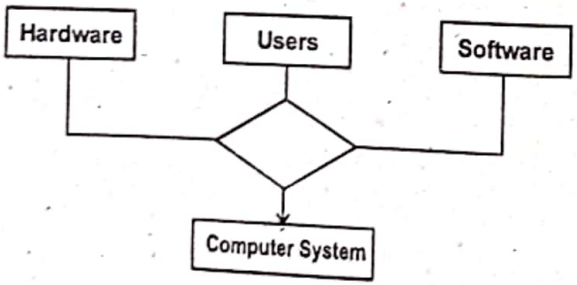


Figure 7.16: Associative data objects

ERD also provides a mechanism that represents the associatively between objects. For example, the data objects (sub-systems) "computer hardware", "computer software", and "users" are associated with the computer system as shown in figure 7.16.

Data modeling and the ERD provide the analyst with a clear and brief or shortcut notation for analyzing data in the data processing system. Therefore, data modeling is used to create analysis models and to design databases.

7.6 DATA FLOW DIAGRAMS (DFDs)

A data flow diagram (DFD) is a graphical representation of the flow of data (or processing steps) through a system or process. A data flow diagram takes an input-process-output view of a system. That is, data objects flow into the system, are transformed by processing elements, and resultant data objects flow out of the system. In DFD, data objects are represented by labeled arrows, and transformations are represented by circles (also called *bubbles*). Figure 7.17 illustrates the input-process-output view of a system.

Figure 7.17: Input-Process-Output view of a system

The purpose of a DFD is to review the flow of data through an information system. It can be used as a communications tool between a system analyst and any person who works in the information system. Thus, the data flow diagram (DFD) is the most common tool for designing database systems.

UML does not support data-flow diagrams (DFDs). The UML activity diagrams are similar to data-flow diagrams (DFDs). So, UML activity diagrams are used to represent processing steps or the flow of data in a system.

Difference between DFD and Flowchart

A DFD is not a flowchart. The main difference between DFD and a flowchart is that DFD represents the flow of data in a system at various levels, while a flowchart represents the flow of control in a program (or represents a sequence of steps to solve a problem). A DFD has no control flow, there are no decision points or rules and no loops.

7.6.1 Applications of DFD

Following are some important applications of data flow diagrams (DFDs):

i) System Analysis

A DFD is commonly used in system analysis. It helps to understand how data enters and leaves the system, how data is used, and where data is stored.

ii) Creating a Business Model

Data flow diagrams can be used to provide a clear representation of any business function. The technique starts with an overall picture of the business (known as a *context diagram*) and continues by analyzing each of the functional areas of interest. The result is a series of diagrams that represent business activities. So a business model consists of one or more data flow diagrams (also known as *business process diagrams*).

iii) Decision Making

A DFD can help for making better decisions by providing an overall visual representation of the system.

iv) Database Designing

A data flow diagram can help in database design. A DFD is drawn before database designing. It provides the conceptual and physical structure of the database that can later be used.

7.6.2 Symbols for DFD

Some important symbols that are commonly used for creating DFD are as follows:

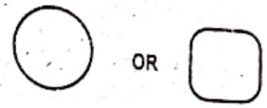
i) Data Flow

Data flow is a pipeline through which packets of data flow. In DFD, data flow is represented by an arrowhead line. It shows the path (or route) for data to move from one part of the system to another. The data flow line is labeled with the name of the data that moves through it.

ii) Process

A process is a part of a system that receives input data, manipulates it, and produces output. It is used to represent any action or task that is performed by the system on the input data. It transforms input data into output. In DFD, a process

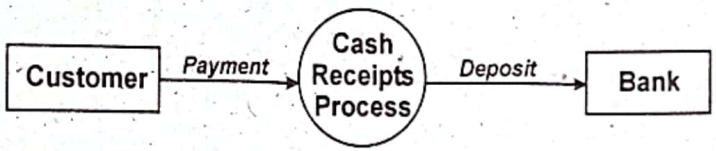
is usually represented by a circle or rounded-edged rectangle. Mostly, a circle symbol is used in DFDs. This symbol is named in one word or short sentence.



iii) External Entity

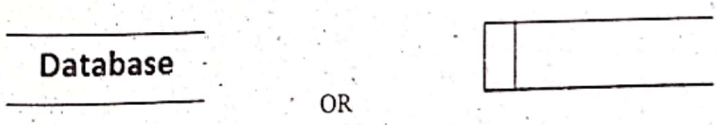
An object that communicates (interacts) with the system is called an *external entity*. An external entity is also called the *terminator*. An external entity is outside the system. It can be a person, another system, or a subsystem. It is a source or destination of data flow. In other words, an external entity represents where certain data comes or goes. In DFD, an external entity is denoted by a rectangle.

The following diagram (context diagram) describes data flow into and out of the system:



iv) Data Store

Data store is a place where data is stored in a system. It is a repository for the storage of data. Data store is also called a *warehouse* or *file* or *database*. It can also be an input or output of a process as well. In DFD, a Data Store is denoted by two horizontal lines between which the Data Store, File, or Database is written. Data store is also denoted by an open-ended narrow rectangle.



v) Collector

The collector symbol is used in a location of the DFD where multiple data flows are combined into a single data flow. In the following figures, figure (a) shows

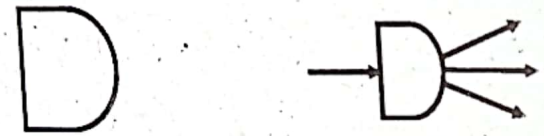
the collector symbol whereas figure (b) shows the collector symbol which combines three data flows into a single data flow.

(a) Collector Symbol

(b) Collector Symbol Combining three data flows into a single data flow

vi) Separator

The separator symbol is used in a location of the DFD where a single data flow is separated into multiple data flows. In the following figures, figure (a) shows the separator symbol whereas figure (b) shows the separator symbol which separates a single data flow into three data flows.



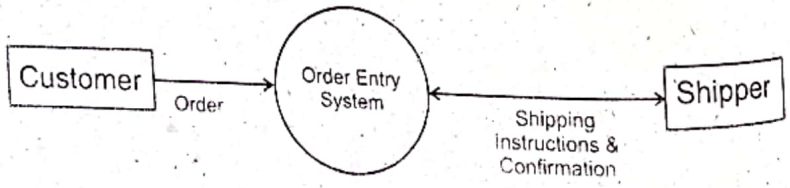
(a) Separator Symbol

(b) Separator Symbol separating a single data flow into three data flows

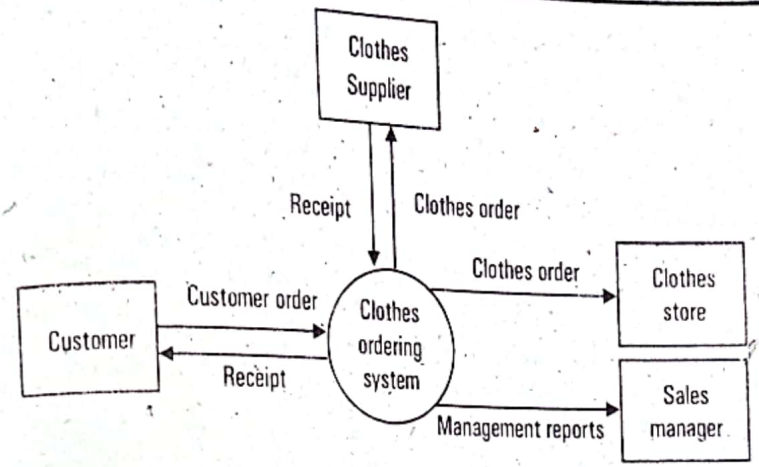
Examples of Data Flow Diagrams

Following are some examples of data flow diagrams (DFDs):

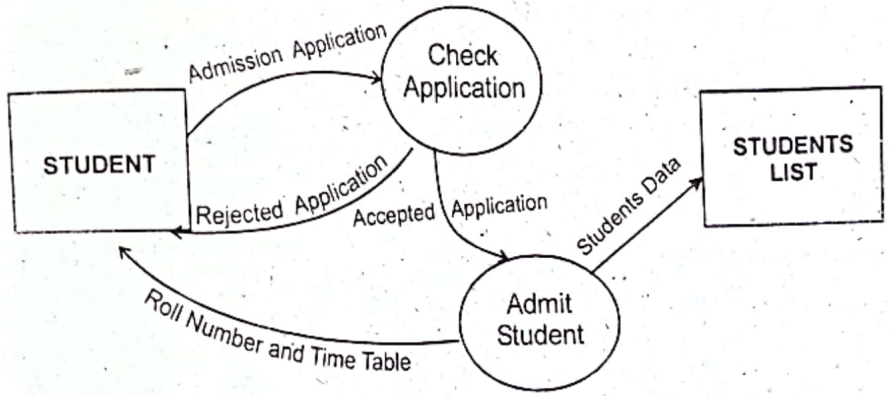
1- The following DFD shows an order entry system that interacts with customers and shippers.



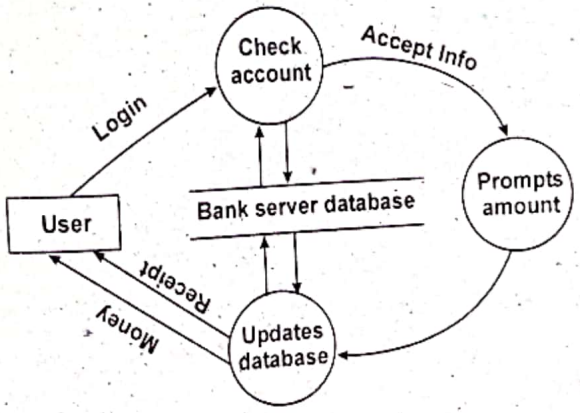
2- The following DFD shows the Clothes Ordering System:



3- The following DFD shows an admission system of a college:



4- The following DFD shows the working of an ATM system in a bank:



7.6.3 Levels of DFD or Types of DFD

A DFD can be divided into different levels. Each level provides more details than the previous level. Different levels of DFD are as follows:

7.6.3.1 Level 0 DFD

Level 0 DFD is also known as a *context diagram*. It is a top-level DFD. It provides a basic overview of the whole system being analyzed or modeled. It contains only a single process (bubble) that represents the whole system and different external entities that interact with the system. The data stores are not shown in Level 0 DFD. Figure 7.18 shows a context diagram for the "Hotel Reservation". This diagram only describes data flow into and out of the process. This level of DFD does not show details of the processing activities of the system.

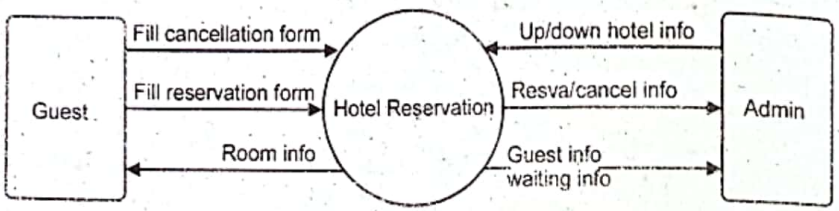


Figure 7.18: Context Diagram of Hotel Reservation

7.6.3.2 Level 1 DFD

Level 1 DFD provides a more detailed view of the system than Level 0 DFD or context diagram. In this level of DFD, the context diagram is decomposed into multiple processes (or bubbles). It means that the single process of the context diagram is broken down into multiple sub-processes. In this level of DFD, detail is given to show which processes are responsible for accepting different inputs and producing outputs. The data stores are also shown in Level 1 DFD. Figure 7.19 shows a Level 1 DFD for the "Hotel Reservation".

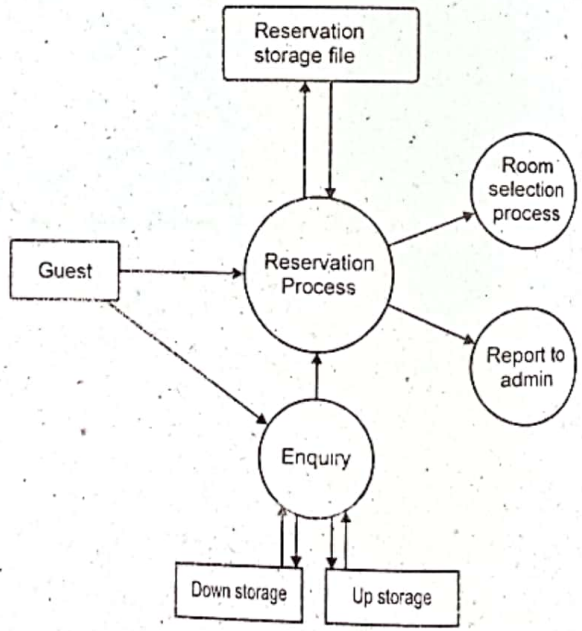


Figure 7.19: Level 1 DFD of Hotel Reservation

7.6.3.3 Level 2 DFD

Level 2 DFD provides a more detailed view of the system than Level 1 DFD. The complicated processes that appear in Level 1 DFD are further divided into sub-processes. Level 2 DFD offers a more detailed look at the processes that make up an information system. It can be used to plan or record the specific makeup of a system.

7.6.4 Advantages and Disadvantages of DFD

Following are some advantages of DFD:

- ★ A DFD is simple and hides the complexities of the system. It is easier to understand by technical and non-technical persons.
- ★ It can provide a detailed representation of system components.
- ★ It gives a functional overview of the system.
- ★ It helps in describing the system scope and its boundaries.
- ★ It is helpful for communicating existing system knowledge to the users.

Following are some disadvantages or limitations of DFD:

- ➢ It only shows the flow of data through a system and does not provide decision points (or decision-making actions).
- ➢ It does not provide a complete picture of the system and leaves some physical entities.

7.7 DATA DICTIONARY

When a data model of a system is derived, many named entities, relationships, functions, etc. are identified. The names given to the entities should be chosen to give the reader a proper meaning. Therefore, it is necessary to provide an organized approach for representing the characteristics of each data object and control item. This can be achieved with the data dictionary. It is defined as:

> "The data dictionary is an organized listing of all data elements relevant to the system, arranged alphabetically, with precise, exact definitions so that both user and system analyst will have a common understanding of inputs, outputs, processes, etc."

The data dictionary is also known as a *data repository* or *system catalog*. It lies at the core or center of the model and contains the descriptions of all data objects consumed or produced by the system.

The data dictionary is implemented as part of a CASE and "Structured Analysis Design" tool. The format of the data dictionary may vary from tool to tool. The most of tools contain the following information:

Name The primary name of the data or control item.

Alias	Second name used for the same data or control item.
Where used / How used	It represents the processes that use the data or control item and how it is used. For example, input to the process and output from the process.
Content Description	It is a notation for representing contents.
Supplementary Information	It represents the other information about data types, limitations, etc.

The data dictionary should include a description of the named entity. If the name represents a composite object, there may be a description of the composition. The other information such as the date of creation, the creator, and representation of the entity may also be included depending on the type of model which is being developed.

All names of entities, types, relations, attributes, or services should be entered in the dictionary. The support software should be available to create and maintain the data dictionary automatically. Once entries in the dictionary are defined, entity-relationship diagrams can be created and object hierarchies can be developed.

To explain the use of the data dictionary and the content description, the data item "telephone number" is specified as input. The telephone number is 8 digits long with 3 digits as the area code. The data dictionary provides an exact definition of the telephone number for the DFD in question. It indicates where and how this data item is used and any supplementary information that is relevant to it. The data dictionary entry for this data item is given as under:

Name	telephone number
Alias	none
Where used/how used	assess against setup (output) dial number (input).
Description	telephone number = [area code | outside number].

The above content description may be read: telephone number is composed of either an "area code" or an "outside number". The "area code" and "outside number" represent composite data and must be refined further in other content description statements.

For large computer-based systems, the data dictionary grows rapidly in size and complexity. In fact, it is extremely difficult to maintain a dictionary manually. For this reason, CASE tools should be used.

7.8 DECISION TREES

Decision trees are a graphical method for defining complex relationships by describing decisions and avoiding problems in communication. A decision tree is a diagram that shows alternative actions and conditions within a horizontal tree framework. Thus, it represents which conditions to consider first, second, and so on.

Decision trees represent the relationship of each condition and their permissible actions. A square node indicates an action and a circle indicates a condition. It forces analysts to consider the sequence of decisions and identifies the actual decision that must be made. Figure 7.20 shows the structure of the decision tree.

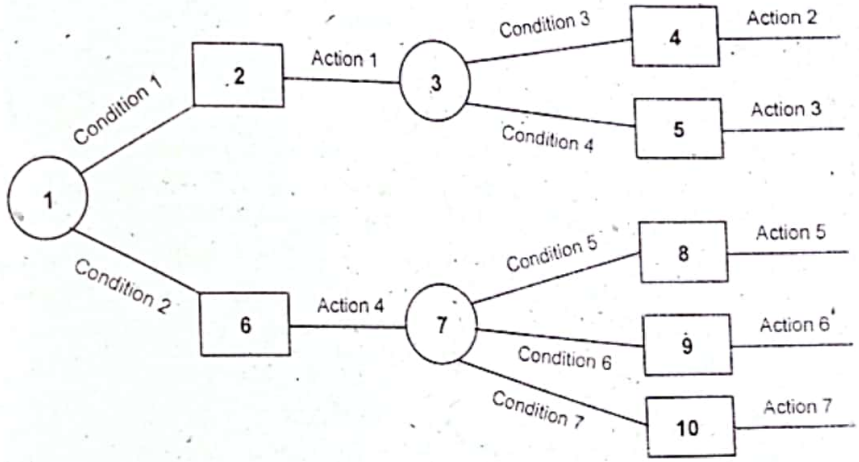


Figure 7.20: The structure of the decision tree

Note that an example of Decision Tree is already given in chapter 5 (see figure 5.2).

7.9 DECISION TABLE

A decision table is a tabular form that presents a set of conditions and their corresponding actions (described in a use case). It is a way to model complicated logic. Like '*if-then-else*' and '*switch-case*' statements, a decision table associates conditions with actions to perform. But, unlike the control structures (such as '*if-then-else*' and '*switch-case*' statements) found in traditional programming languages, a decision table can associate many independent conditions with several actions in an elegant way.

7.9.1 Structure of Decision Table

A decision table is typically divided into four quadrants or sections as shown below.

The four quadrants

Conditions	**Condition Alternatives**
Actions	**Action entries**

The upper left-hand quadrant contains a list of conditions. The upper right-hand quadrant contains a list of condition rules for alternatives. The lower left-hand quadrant contains a list of all actions that are possible based on combinations of conditions. The right-hand quadrants form a matrix that indicates condition combinations and the corresponding actions that will occur for a specific combination. Therefore, each column of the matrix may be interpreted as a *processing rule*.

7.9.2 Developing Decision Table

In order to build a decision table, we need to determine the maximum size of the table, eliminate any impossible situations, redundancies, and simplify the table as much as possible. The following steps provide some guidelines for developing a decision table:

1- Determine the number of conditions that may affect the decision. Combine rows that overlap such as conditions that are mutually exclusive. The number of conditions becomes the number of rows in the top half of the decision table.

2- Determine the number of possible actions that can be taken. This becomes the number of rows in the lower half of the decision table.

3- Determine the number of condition alternatives for each condition. In the simplest form of a decision table, there would be two alternatives (Y or N) for each condition. In an extended-entry table, there may be many alternatives for each condition.

4- Calculate the maximum number of columns in the decision table by multiplying the number of alternatives for each condition. If there were four conditions and two alternatives (Y or N) for each of the conditions, there would be sixteen possibilities as follows:

Condition 1:	x	2 Alternatives
Condition 2:	x	2 Alternatives
Condition 3:	x	2 Alternatives
Condition 4:	x	2 Alternatives

16 Possibilities

5- Fill in the Condition Alternatives. Start with the first condition and divide the number of columns by the number of alternatives for that condition. In the foregoing example, there are sixteen columns and two alternatives (Y and N), so sixteen divided by two is eight (16/2 = 8). Then choose one of the alternatives and write Y in all of the eight columns. Finish by writing N in the remaining eight columns as follows:

Condition 1	Y	Y	Y	Y	Y	Y	Y	Y	N	N	N	N	N	N	N	N

Repeat this for each condition using a subset of the table:

Condition 1	Y	Y	Y	Y	Y	Y	Y	Y	N	N	N	N	N	N	N	N
Condition 2	Y	Y	Y	Y	N	N	N	N								
Condition 3	Y	Y	N	N												
Condition 4	Y	N														

Continue the pattern for each condition:

Condition 1	Y	Y	Y	Y	Y	Y	Y	Y	N	N	N	N	N	N	N	N
Condition 2	Y	Y	Y	Y	N	N	N	N	Y	Y	Y	Y	N	N	N	N
Condition 3	Y	Y	N	N	Y	Y	N	N	Y	Y	N	N	Y	Y	N	N
Condition 4	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N

6- Complete the table by inserting an X where rules suggest certain actions.

7- Combine rules where it is apparent that an alternative does not make a difference in the outcome, for example:

Condition 1	Y Y
Condition 2	Y N
Action 1	X X

can be expressed as:

Condition 1	Y
Condition 2	-
Action 1	X

The dash (-) signifies that condition 2 can be either Y or N and action will still be taken.

8- Check the table for any impossible situations, contradictions, and redundancies.

9- Rearrange the conditions and actions (or even rules) to make the decision table more understandable.

Example 1: Printer Troubleshooter

A technical support company writes a decision table to diagnose printer problems based upon symptoms described to them over the phone from their clients.

Printer Troubleshooter									
		Rules							
Conditions	Printer does not print	Y	Y	Y	Y	N	N	N	N
	A red light is flashing	Y	Y	N	N	Y	Y	N	N
	Printer is unrecognized	Y	N	Y	N	Y	N	Y	N
Actions	Check the power cable			X					
	Check the printer-computer cable	X		X					
	Ensure printer software is installed	X		X		X		X	
	Check/replace ink	X	X			X	X		
	Check for paper jam		X		X				

Of course, this is just a simple example (and it does not necessarily correspond to the reality of printer troubleshooting), but even so, it demonstrates how decision tables can scale to several conditions with many possibilities.

Example 2: Public Utility Billing System

If the customer account is billed using a fixed rate method, a minimum monthly charge is accessed for consumption of less than 100 kWh. Otherwise, computer billing applies a schedule "A" rate structure. However, if the account is billed by using a variable rate method, a schedule "A" rate structure will apply to consumption below 100 KWH, with additional consumption billed according to schedule "B".

Rules					
Conditions	1	2	3	4	5
Fixed Rate	T	T	F	F	F
Variable Rate	F	F	T	T	F
Consumption <100 kwh	T	F	T	F	

Consumption >=100 kwh	F	T	F	T	
Actions					
Min. Monthly Charge	√				
Schedule A billing		√	√		
Schedule B billing				√	
Other Treatment					√

7.10 Program Design Language

Program design language (PDL) is also called *structured English* or *pseudocode*. It includes the logical structure of a programming language which gives a more understandable and precise description of the process. It is based on procedural logic that uses construction and imperative sentences designed to perform an operation for action.

- It is best used when sequences and loops in a program must be considered and the problem needs sequences of actions with decisions.
- It does not have strict syntax rules. It expresses all logic in terms of sequential decision structures and iterations.

For example, see the following sequence of actions:

```
If customer pays advance Then
    Give 5% Discount
Else
    If purchase amount >= 10,000 Then
        If the customer is a regular customer Then
            Give 5% Discount
        Else
            No Discount
        End If
    Else
        No Discount
    End If
End If
```

Short Answers to the Questions

Q.1 What is meant by software requirements analysis?

Software requirements analysis simply means examining (analyzing or studying), and describing software requirements so that requirements that are genuine and needed can be fulfilled to solve the problem.

Q.2 What is UML?

UML stands for Unified Modeling Language (UML). It is a general-purpose standard graphical modeling language in the field of software engineering. The main aim of UML is to assist in software development.

Q.3 What is UML activity diagram?

This diagram shows the flow of activities described by the use case (or activities that are involved in a process or data processing). An activity diagram is similar to a DFD (data-flow diagram).

Q.4 What is case diagram?

This diagram shows the interactions between a system and its environment, i.e. external actors (users or other systems/devices). A use case can be taken as a simple scenario that describes what a user expects from a system. Each use case represents a separate task that involves external interaction with a system.

Q.5 What is class diagram?

This diagram shows the object classes in the system and the associations between these classes. An association is a link between classes that indicates a relationship between these classes. The purpose of the class diagram is to represent the static view of the system.

Q.6 What is state diagram?

This diagram shows how the system reacts to internal and external events. The state diagram also shows system states and events that cause transitions from one state to another.

Q.7 What is analysis model?

An analysis model is a technical representation of the system or software to be built. It means that the analysis model represents the information, functions, and behavior of the software system to be built. Afterward, the analysis model is translated into the architecture, interface, and component-level design in the design modeling.

Q.8 What are scenario-based elements?

Scenario-based elements of the requirements model are often the first part of the model that is developed. So, they serve as input for the creation of other modeling elements. Scenario-based elements consist of use cases in text form, use-case diagrams, activity diagrams, and swimlane diagrams.

Q.9 What are class-based elements?

A class is a collection of things that have similar attributes and common behaviors. Class-based elements consist of class diagrams, analysis packages, CRC models, and collaboration diagrams.

Q.10 What is an analysis package?

Categorization is an important part of analysis modeling. It means that various elements of the analysis model (e.g., use cases, analysis classes) are categorized in a manner that packages them as a grouping. It is given a representative name. This mechanism is called an *analysis package*.

Q.11 What is CRC model?

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, the name of the class is written. In the body of the card, class responsibilities are written on the left and the collaborators are written on the right.

Q.12 What are behavioral elements?

Behavioral elements represent the states of the system and how they are changed by internal and external events. Behavioral elements consist of state diagrams and sequence diagrams.

Q.13 What are flow-oriented elements?

The flow elements are data flow diagrams (DFDs), control flow diagrams (CFDs), and processing narratives.

Q.14 What is structured analysis?

The structured analysis is a systematic development approach that enables the analyst to understand the overall functionality of the software system and its activities. It considers data and the processes (that manipulate or transform the data) as separate entities. Data objects are modeled in a way that defines the attributes of data objects and relationships among them. Processes that manipulate or transform the data are modeled in a manner that shows how data objects are manipulated/transformed and flow through the system.

Q.15 What is object-oriented analysis?

The object-oriented analysis focuses on the definition of classes and the manner in which they collaborate with one another. OOA is the first technical activity performed as part of object-oriented software engineering. The main purpose of OOA is to identify the objects of a system that have to be implemented. OOA models are developed during the requirement analysis. The OOA does not include detail of the individual objects in a system to be built.

Q.16 Define data modeling.

Data modeling is the process of creating a simplified diagram of a software system and the data elements it contains, to represent the data and how it flows within a system. Data models are created as part of overall requirements modeling.

Q.17 What is data flow diagram?

A data flow diagram (DFD) is a graphical representation of the flow of data (
processing steps) through a system or process. A data flow diagram takes an inpu
process-output view of a system. That is, data objects flow into the system, a
transformed by processing elements, and resultant data objects flow out of the system.
DFD, data objects are represented by labeled arrows, and transformations are represent
by circles (also called *bubbles*).

Q.18 What is level-0 DFD?

Level 0 DFD is also known as a *context diagram*. It is a top-level DFD. It provides
basic overview of the whole system being analyzed or modeled. It contains only a sing
process (bubble) that represents the whole system and different external entities th
interact with the system. The data stores are not shown in Level 0 DFD.

Multiple Choice Questions – MCQs

01) **The requirements model refers _____.**

(a) technical representation of a system (b) a set of models
(c) abstract view of proposed system (d) All of these

02) **Requirements modeling are combination of _____.**

(a) Text (b) Diagrammatic form
(c) Resources and technical representation (d) All of these

03) **Amongst which of the following is responsible to do th**

(a) Software engineer

01 (d)			04 (d)	05 (d)	06 (a)	07 (a)	08 (a)	09 (d)	10 (d)
11 (c)	12 (c)	13 (c)	14 (d)	15 (d)	16 (d)	17 (a)	18 (d)	19 (b)	20 (a)
21 (d)	22 (b)	23 (c)	24 (d)	25 (d)	26 (a)	27 (b)	28 (a)	29 (d)	

EXERCISE

Q#1 **Explain in detail requirements analysis.**

Q#2 **What is UML? Briefly describe any five UML diagrams.**

Q#3 **What do you mean by analysis model and discuss its elements?**

Q#4 **How is data modeling implemented in a system and what are its basic elements.**

Q#5 What is ERD? Differentiate between cardinality and modality.

Q#6 Suggest why data-flow diagrams are intuitive and easy to understand by non-technical staff.

Q#7 Model the data processing, which might take place in an electronic mail system that can send and receive a message from remote computers.

Q#8 Develop an E-R diagram that will describe data objects, relationships, and attributes for the systems listed in the problem.

Q#9 Describe the following terms:

(a) Data objects

(b) Attributes

(c) Relationships

(d) Domain

(e) Data Dictionary

Q#10 Explain the Data Flow Diagram (DFD).