

Software Engineering

Module 2

Requirements Analysis and Modelling

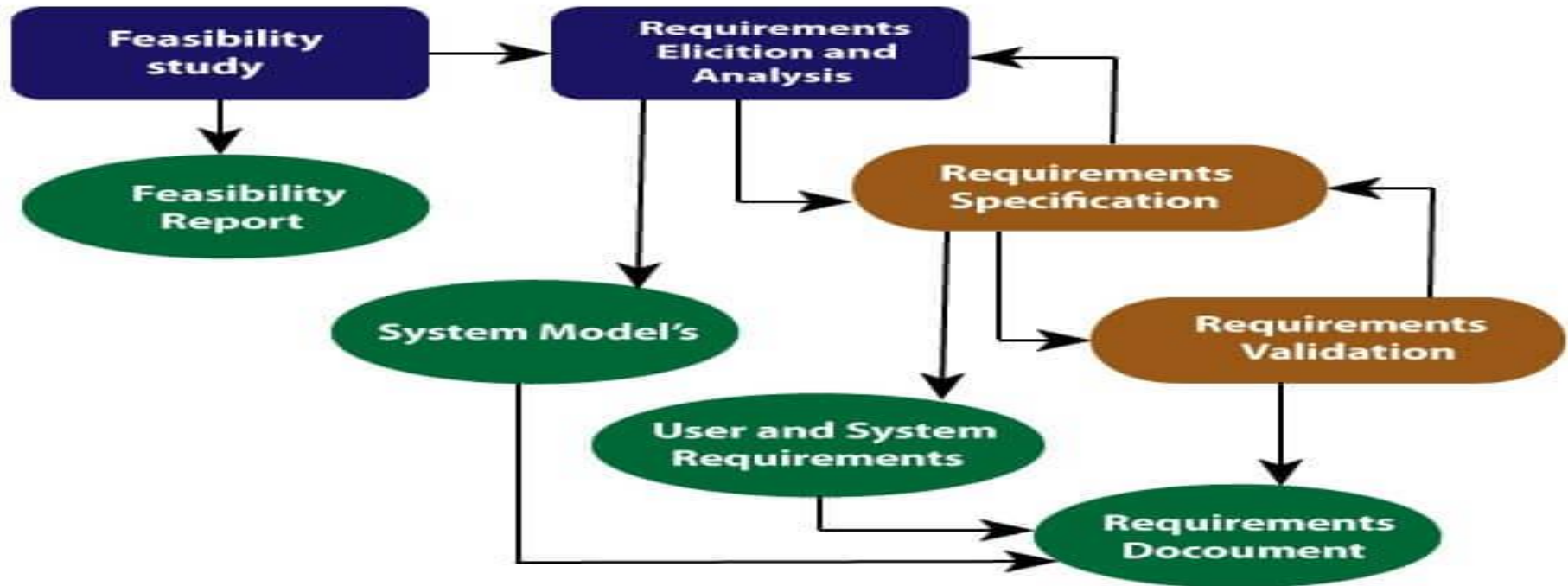
**By,
Ms.Swati Mahalle**

Requirement Engineering

- **Requirements engineering (RE)** refers to the process of defining, documenting, and maintaining requirements in the engineering design process.
- Requirement engineering provides the appropriate mechanism to understand what the customer desires, analyzing the need, and assessing feasibility, negotiating a reasonable solution, specifying the solution clearly, validating the specifications and managing the requirements as they are transformed into a working system.
- Thus, requirement engineering is the disciplined application of proven principles, methods, tools, and notation to describe a proposed system's intended behavior and its associated constraints.

Requirement Engineering Process

- It is a four-step process, which includes -
- Feasibility Study
- Requirement Elicitation and Analysis
- Software Requirement Specification
- Software Requirement Validation
- Software Requirement Management



Requirement Engineering Process

Requirement Engineering Process

1. Feasibility Study:

- The objective behind the feasibility study is to create the reasons for developing the software that is acceptable to users, flexible to change and conformable to established standards.

Types of Feasibility:

- **Technical Feasibility** - Technical feasibility evaluates the current technologies, which are needed to accomplish customer requirements within the time and budget.
- **Operational Feasibility** - Operational feasibility assesses the range in which the required software performs a series of levels to solve business problems and customer requirements.
- **Economic Feasibility** - Economic feasibility decides whether the necessary software can generate financial profits for an organization.

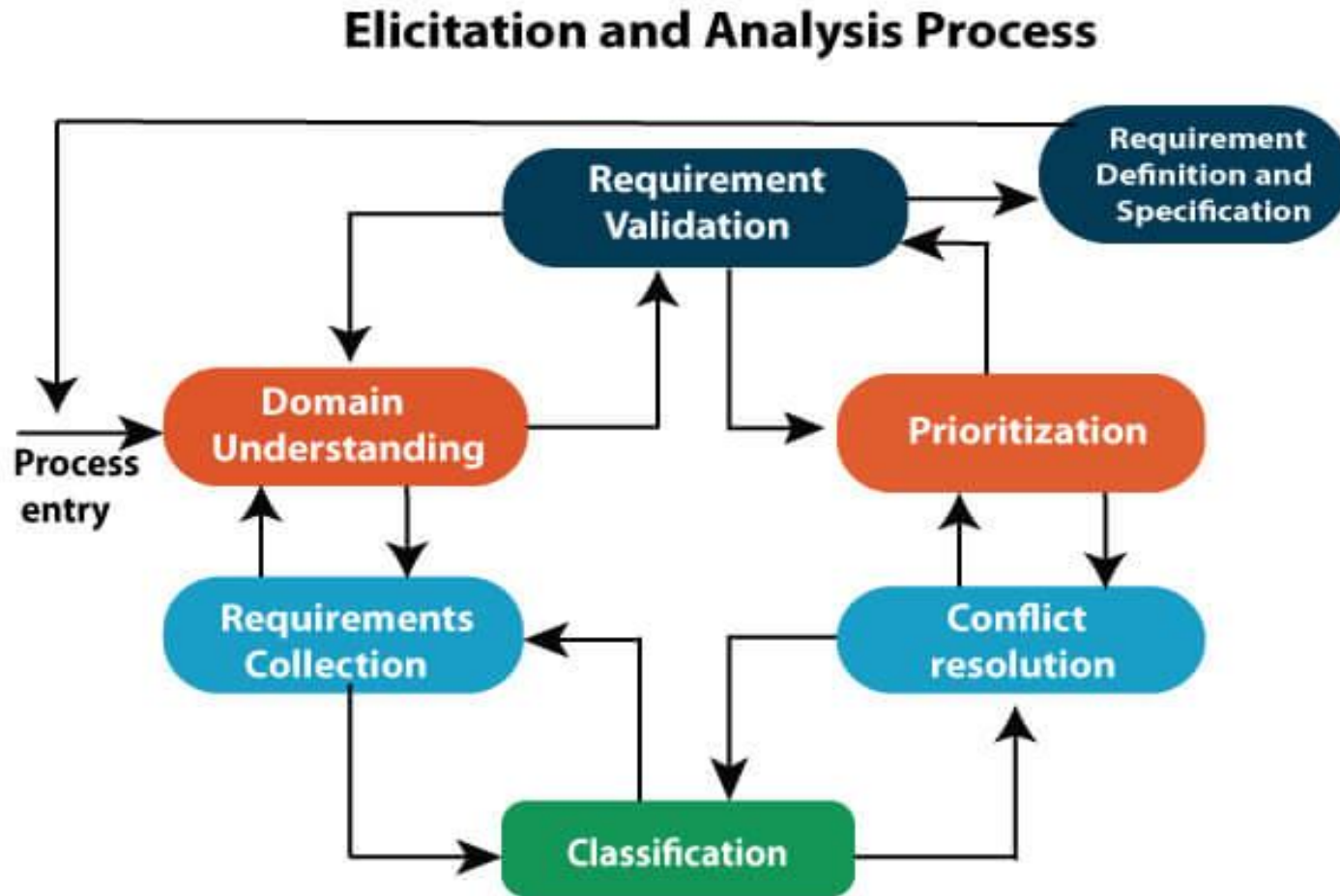
2. Requirement Elicitation and Analysis:

- This is also known as the **gathering of requirements**. Here, requirements are identified with the help of customers and existing systems processes, if available.
- Analysis of requirements starts with requirement elicitation. The requirements are analyzed to identify inconsistencies, defects, omission, etc. We describe requirements in terms of relationships and also resolve conflicts if any.

Problems of Elicitation and Analysis

- Getting all, and only, the right people involved.
- Stakeholders often don't know what they want
- Stakeholders express requirements in their terms.
- Stakeholders may have conflicting requirements.
- Requirement change during the analysis process.
- Organizational and political factors may influence system requirements.

2. Requirement Elicitation and Analysis:



3. Software Requirement Specification:

- Software requirement specification is a kind of document which is created by a software analyst after the requirements collected from the various sources - the requirement received by the customer written in ordinary language.
- It is the job of the analyst to write the requirement in technical language so that they can be understood and beneficial by the development team.

The models used at this stage include ER diagrams, data flow diagrams (DFDs), function decomposition diagrams (FDDs), data dictionaries, etc.

- **Data Flow Diagrams:** Data Flow Diagrams (DFDs) are used widely for modeling the requirements. DFD shows the flow of data through a system. The system may be a company, an organization, a set of procedures, a computer hardware system, a software system, or any combination of the preceding. The DFD is also known as a data flow graph or bubble chart.
- **Data Dictionaries:** Data Dictionaries are simply repositories to store information about all data items defined in DFDs. At the requirements stage, the data dictionary should at least define customer data items, to ensure that the customer and developers use the same definition and terminologies.
- **Entity-Relationship Diagrams:** Another tool for requirement specification is the entity-relationship diagram, often called an "*E-R diagram*." It is a detailed logical representation of the data for the organization and uses three main constructs i.e. data entities, relationships, and their associated attributes.

4. Software Requirement Validation:

- After requirement specifications developed, the requirements discussed in this document are validated. The user might demand illegal, impossible solution or experts may misinterpret the needs. Requirements can be the check against the following conditions –
- If they can practically implement
- If they are correct and as per the functionality and specially of software
- If there are any ambiguities
- If they are full
- If they can describe

Requirements Validation Techniques

- **Requirements reviews/inspections:** systematic manual analysis of the requirements.
- **Prototyping:** Using an executable model of the system to check requirements.
- **Test-case generation:** Developing tests for requirements to check testability.
- **Automated consistency analysis:** checking for the consistency of structured requirements descriptions.

Software Requirement Management:

- Requirement management is the process of managing changing requirements during the requirements engineering process and system development.
- New requirements emerge during the process as business needs a change, and a better understanding of the system is developed.
- The priority of requirements from different viewpoints changes during development process.
- The business and technical environment of the system changes during the development.

Prerequisite of Software requirements

- Collection of software requirements is the basis of the entire software development project. Hence they should be clear, correct, and well-defined.

A complete Software Requirement Specifications should be:

- Clear
- Correct
- Consistent
- Coherent
- Comprehensible
- Modifiable
- Verifiable
- Prioritized
- Unambiguous
- Traceable
- Credible source

Software Requirements: Largely software requirements must be categorized into two categories:

1. **Functional Requirements:** Functional requirements define a function that a system or system element must be qualified to perform and must be documented in different forms. The functional requirements are describing the behavior of the system as it correlates to the system's functionality.
2. **Non-functional Requirements:** This can be the necessities that specify the criteria that can be used to decide the operation instead of specific behaviors of the system.
Non-functional requirements are divided into two main categories:
 - **Execution qualities** like security and usability, which are observable at run time.
 - **Evolution qualities** like testability, maintainability, extensibility, and scalability that embodied in the static structure of the software system.

Software Requirement Specifications

- **Software Requirement Specification (SRS) Format** as the name suggests, is a complete specification and description of requirements of the software that need to be fulfilled for the successful development of the software system. These requirements can be functional as well as non-functional depending upon the type of requirement. The interaction between different customers and contractors is done because it is necessary to fully understand the needs of customers.
- Depending upon information gathered after interaction, SRS is developed which describes requirements of software that may include changes and modifications that is needed to be done to increase quality of product and to satisfy customer's demand.

Introduction

- **Purpose of this Document** – At first, main aim of why this document is necessary and what's purpose of document is explained and described.
- **Scope of this document** – In this, overall working and main objective of document and what value it will provide to customer is described and explained. It also includes a description of development cost and time required.
- **Overview** – In this, description of product is explained. It's simply summary or overall review of product.

General description

- In this, general functions of product which includes objective of user, a user characteristic, features, benefits, about why its importance is mentioned. It also describes features of user community.

Functional Requirements

- In this, possible outcome of software system which includes effects due to operation of program is fully explained. All functional requirements which may include calculations, data processing, etc. are placed in a ranked order. Functional requirements specify the expected behavior of the system-which outputs should be produced from the given inputs. They describe the relationship between the input and output of the system. For each functional requirement, detailed description all the data inputs and their source, the units of measure, and the range of valid inputs must be specified.

Interface Requirements

- In this, software interfaces which mean how software program communicates with each other or users either in form of any language, code, or message are fully described and explained. Examples can be shared memory, data streams, etc.

Performance Requirements

- In this, how a software system performs desired functions under specific condition is explained. It also explains required time, required memory, maximum error rate, etc. The performance requirements part of an SRS specifies the performance constraints on the software system. All the requirements relating to the performance characteristics of the system must be clearly specified. There are two types of performance requirements: static and dynamic. Static requirements are those that do not impose constraint on the execution characteristics of the system. Dynamic requirements specify constraints on the execution behaviour of the system.

Design Constraints

- In this, constraints which simply means limitation or restriction are specified and explained for design team. Examples may include use of a particular algorithm, hardware and software limitations, etc. There are a number of factors in the client's environment that may restrict the choices of a designer leading to design constraints such factors include standards that must be followed resource limits, operating environment, reliability and security requirements and policies that may have an impact on the design of the system. An SRS should identify and specify all such constraints.

Non-Functional Attributes

- In this, non-functional attributes are explained that are required by software system for better performance. An example may include Security, Portability, Reliability, Reusability, Application compatibility, Data integrity, Scalability capacity, etc.

Preliminary Schedule and Budget

- In this, initial version and budget of project plan are explained which include overall time duration required and overall cost required for development of project.

Appendices

- In this, additional information like references from where information is gathered, definitions of some specific terms, acronyms, abbreviations, etc. are given and explained.

Uses of SRS document

- Development team require it for developing product according to the need.
- Test plans are generated by testing group based on the describe external behaviour.
- Maintenance and support staff need it to understand what the software product is supposed to do.
- Project manager base their plans and estimates of schedule, effort and resources on it.
- customer rely on it to know that product they can expect.
- As a contract between developer and customer.
- in documentation purpose.

Conclusion

- Software development requires a well-structured Software Requirement Specification (SRS). It helps stakeholders communicate, provides a roadmap for development teams, guides testers in creating effective test plans, guides maintenance and support employees, informs project management decisions, and sets customer expectations. The SRS document helps ensure that the software meets functional and non-functional requirements, resulting in a quality product on time and within budget.

Data Flow Diagrams


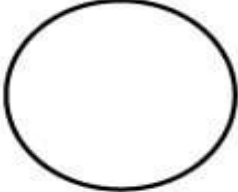

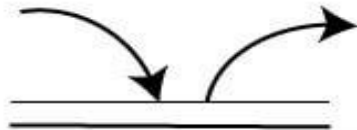
- A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. A neat and clear DFD can depict the right amount of the system requirement graphically. It can be manual, automated, or a combination of both.
- It shows how data enters and leaves the system, what changes the information, and where data is stored.
- The objective of a DFD is to show the scope and boundaries of a system as a whole. It may be used as a communication tool between a system analyst and any person who plays a part in the order that acts as a starting point for redesigning a system. The DFD is also called as a data flow graph or bubble chart.

Data Flow Diagrams

The following observations about DFDs are essential:

- All names should be unique. This makes it easier to refer to elements in the DFD.
- Remember that DFD is not a flow chart. Arrows is a flow chart that represents the order of events; arrows in DFD represents flowing data. A DFD does not involve any order of events.
- Suppress logical decisions. If we ever have the urge to draw a diamond-shaped box in a DFD, suppress that urge! A diamond-shaped box is used in flow charts to represents decision points with multiple exists paths of which the only one is taken. This implies an ordering of events, which makes no sense in a DFD.
- Do not become bogged down with details. Defer error conditions and error handling until the end of the analysis.

Standard symbols for DFDs are derived from the electric circuit diagram analysis and are shown in fig:

Symbol	Name	Function
	Data flow	Used to Connect Processes to each , other , to sources or Sinks; te arrow head indicates direction of data flow.
	Process	Performs Some transformation of Input data to yield output data.
	Source of Sink (External Entity)	A Source of System inputs or Sink of System outputs.
	Data Store	A repository of data; the arrow heads indicate net inputs and net outputs to store.

Symbols for Data Flow Diagrams

Standard symbols for DFDs are derived from the electric circuit diagram analysis

- **Circle:** A circle (bubble) shows a process that transforms data inputs into data outputs.
- **Data Flow:** A curved line shows the flow of data into or out of a process or data store.
- **Data Store:** A set of parallel lines shows a place for the collection of data items. A data store indicates that the data is stored which can be used at a later stage or by the other processes in a different order. The data store can have an element or group of elements.
- **Source or Sink:** Source or Sink is an external entity and acts as a source of system inputs or sink of system outputs.

Levels in Data Flow Diagrams (DFD)

- The DFD may be used to perform a system or software at any level of abstraction. Infact, DFDs may be partitioned into levels that represent increasing information flow and functional detail. Levels in DFD are numbered 0, 1, 2 or beyond. Here, we will see primarily three levels in the data flow diagram, which are: 0-level DFD, 1-level DFD, and 2-level DFD.

1) 0-level DFDM

- It is also known as fundamental system model, or context diagram represents the entire software requirement as a single bubble with input and output data denoted by incoming and outgoing arrows.
- Then the system is decomposed and described as a DFD with multiple bubbles.
- Parts of the system represented by each of these bubbles are then decomposed and documented as more and more detailed DFDs.
- This process may be repeated at as many levels as necessary until the program at hand is well understood.
- It is essential to preserve the number of inputs and outputs between levels, this concept is called leveling by DeMacro.
- Thus, if bubble "A" has two inputs x_1 and x_2 and one output y , then the expanded DFD, that represents "A" should have exactly two external inputs and one external output as shown in fig:

1) 0-level DFD

- The Level-0 DFD, also called context diagram of the result management system is shown in fig. As the bubbles are decomposed into less and less abstract bubbles, the corresponding data flow may also be needed to be decomposed.

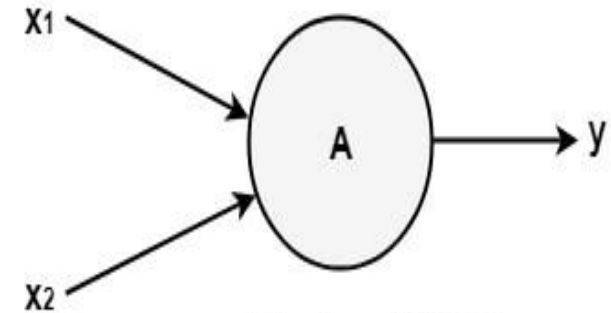


Fig: Level-0 DFD.

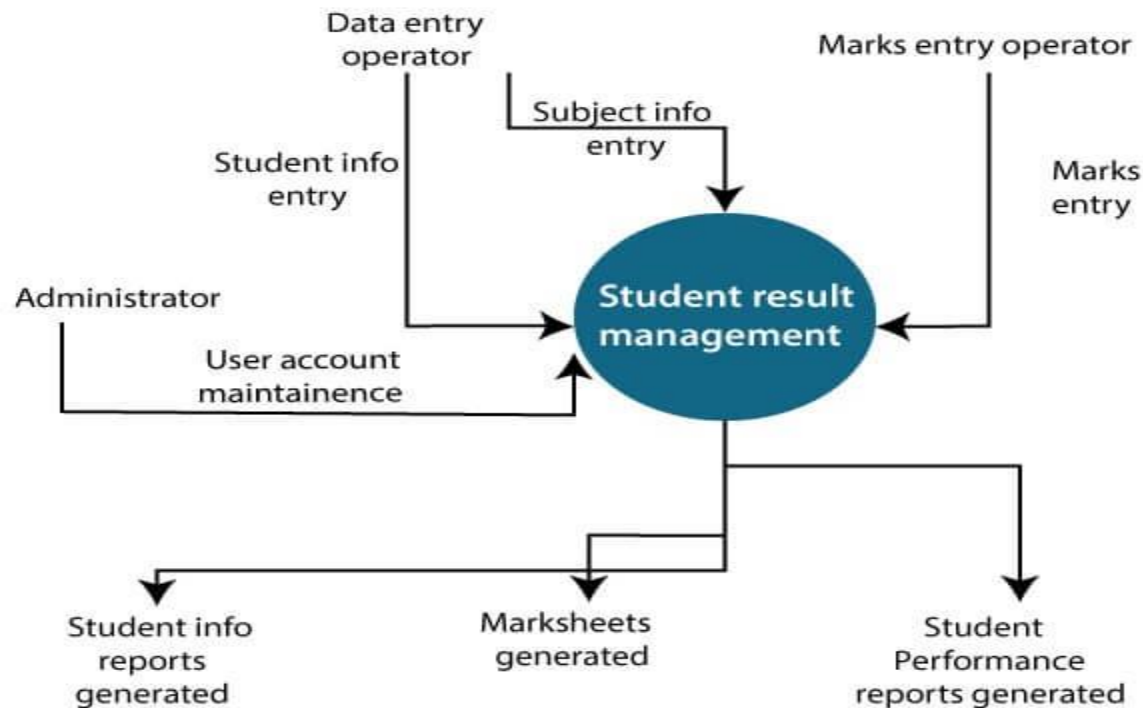


Fig: Level-0 DFD of result management system

• 1-level DFD

- In 1-level DFD, a context diagram is decomposed into multiple bubbles/processes. In this level, we highlight the main objectives of the system and breakdown the high-level process of 0-level DFD into subprocesses.

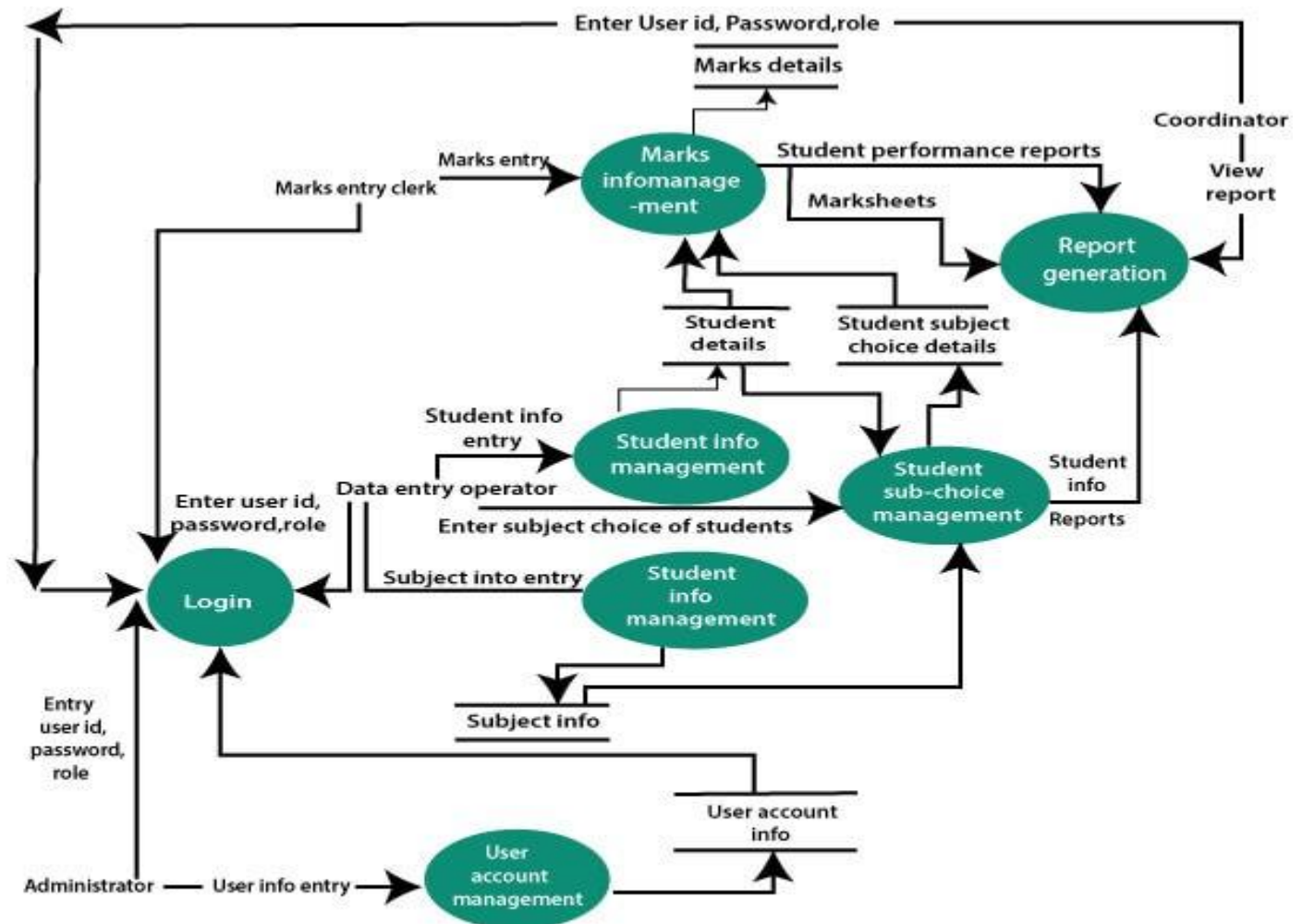
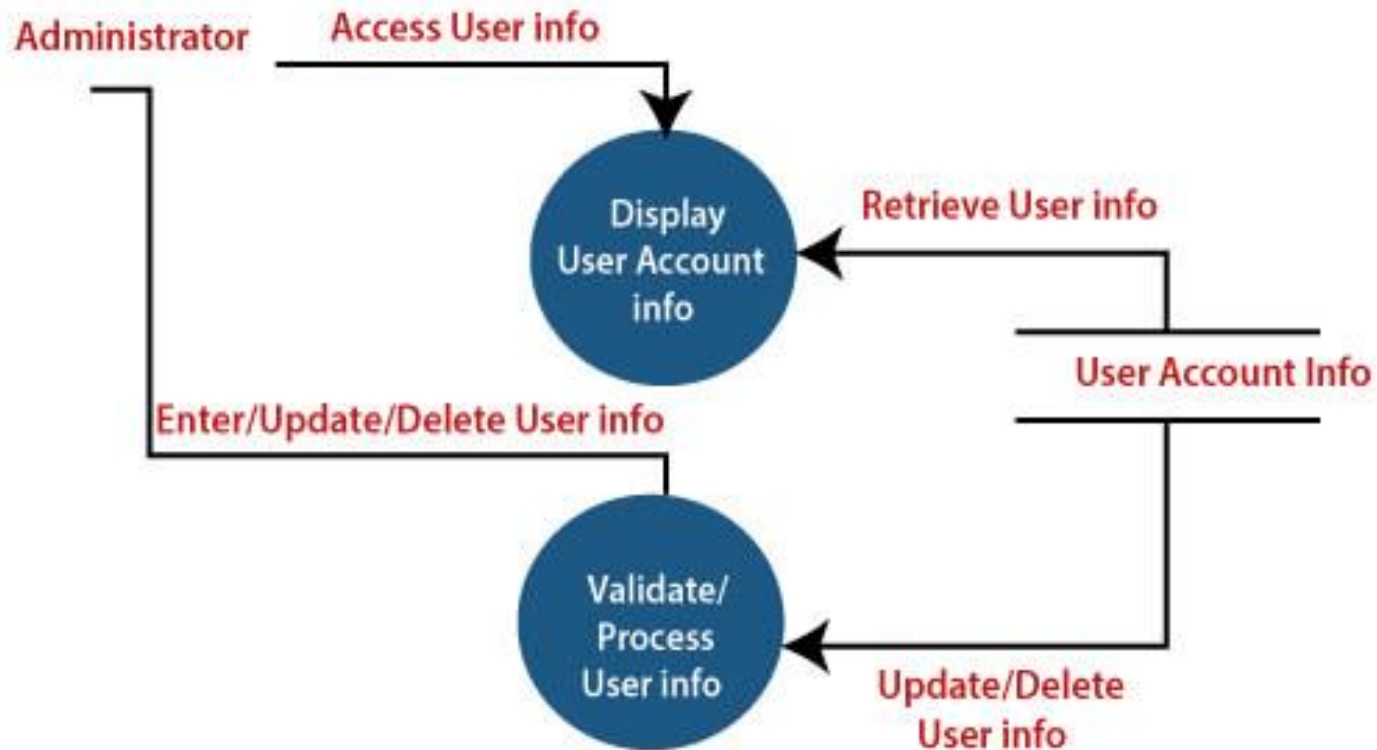


Fig: Level-1 DFD of result management system

2-Level DFD

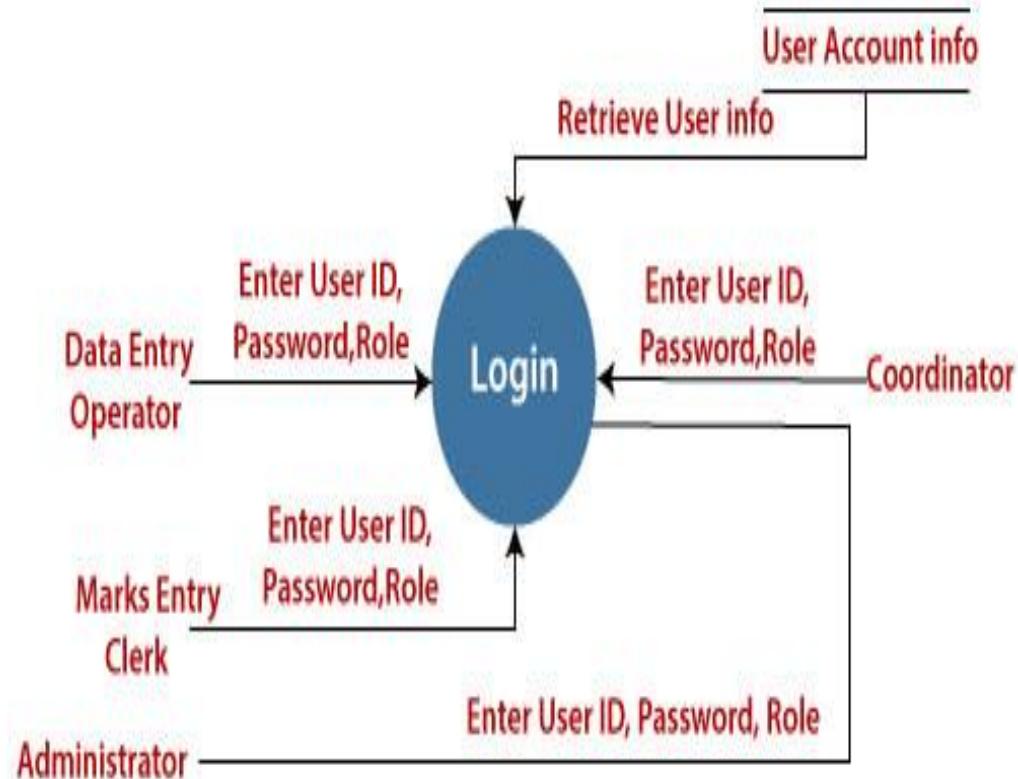
- 2-level DFD goes one process deeper into parts of 1-level DFD. It can be used to project or record the specific/necessary detail about the system's functioning.

1. User Account Maintenance

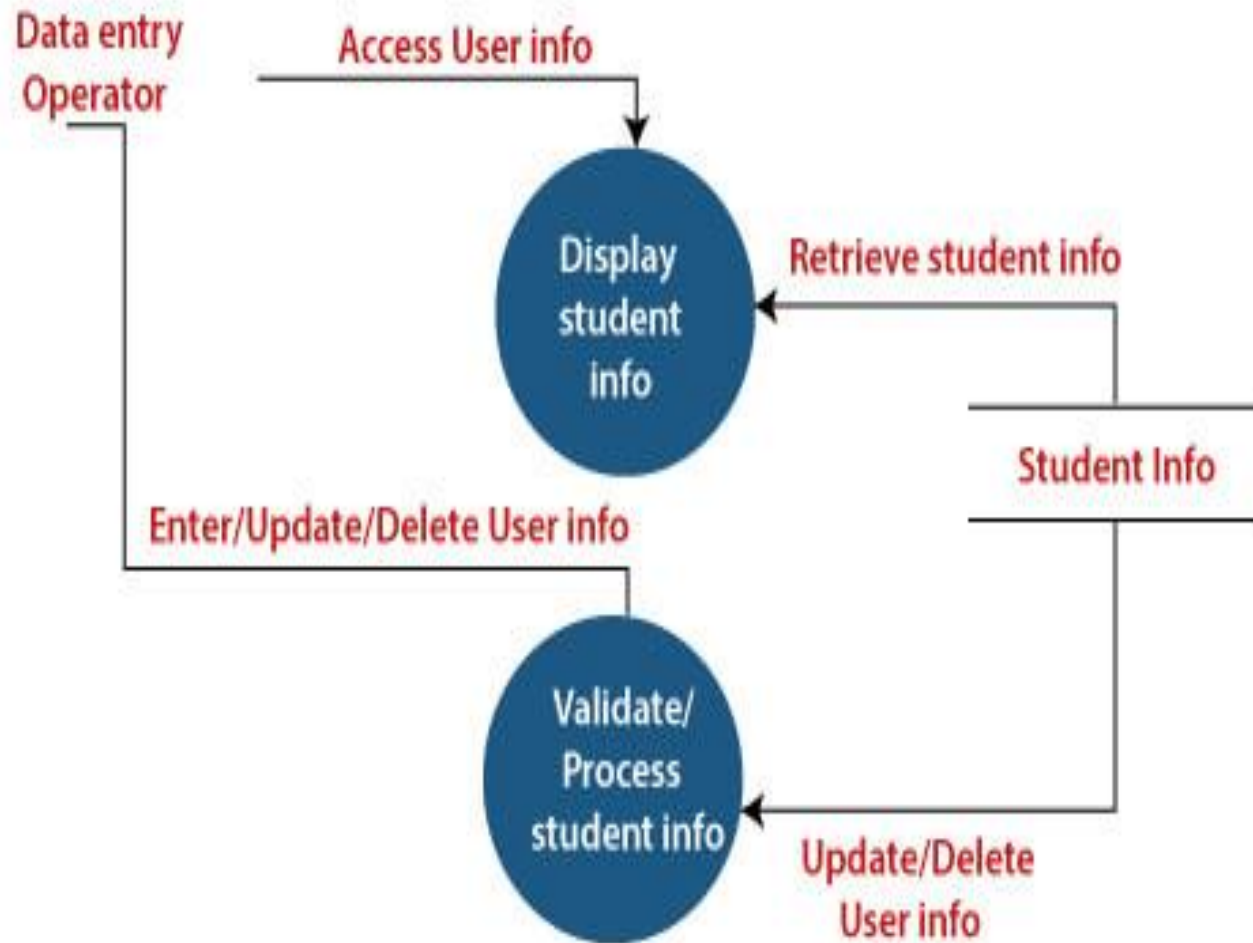


2. Login

The level 2 DFD of this process is given below:

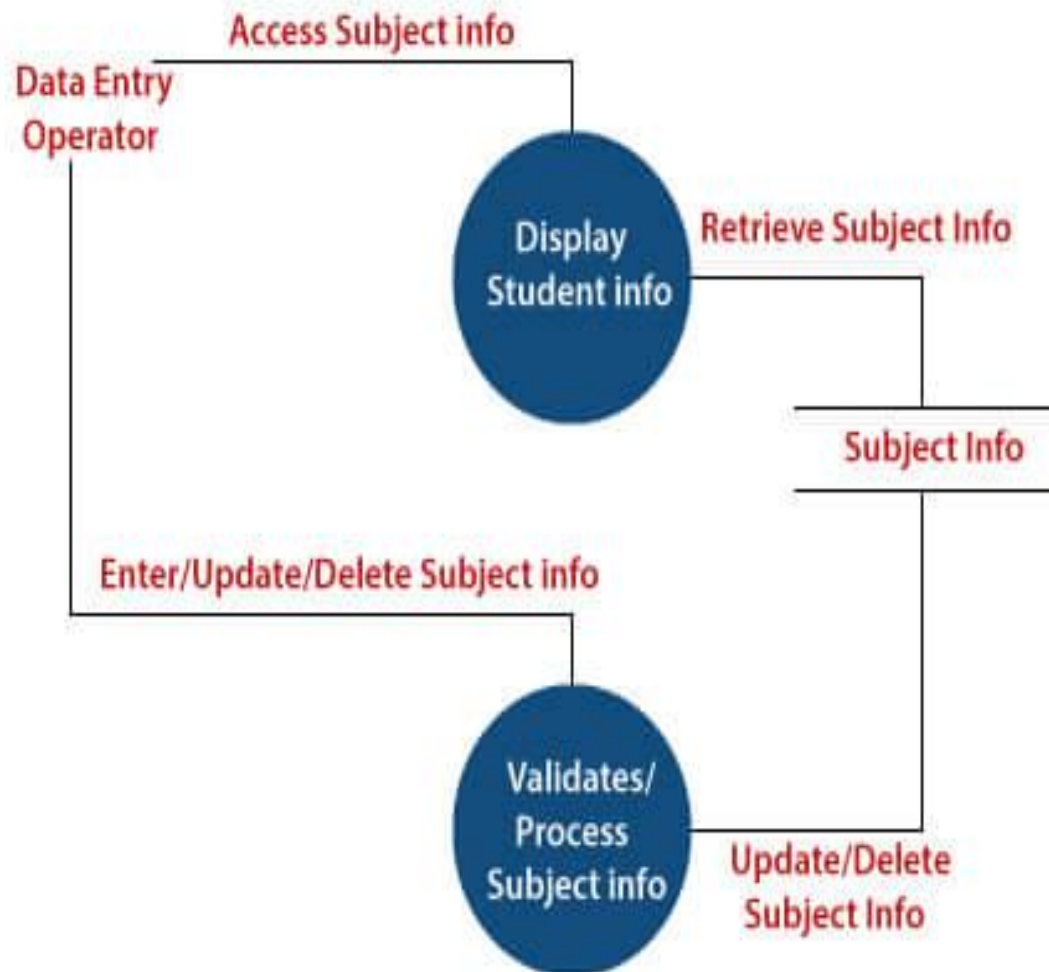


3. Student Information Management



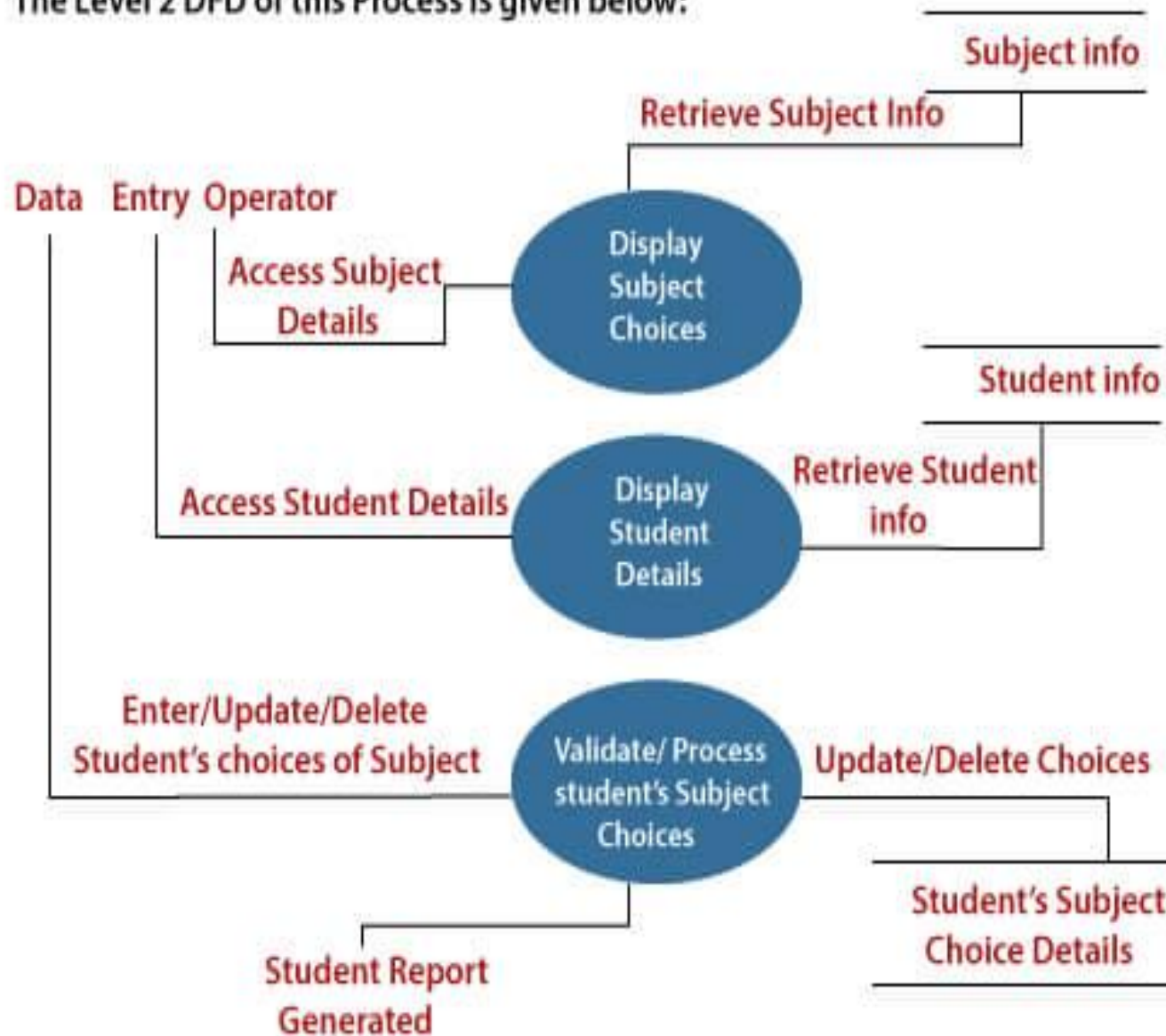
4. Subject Information Management

The level 2 DFD of this process is given below:



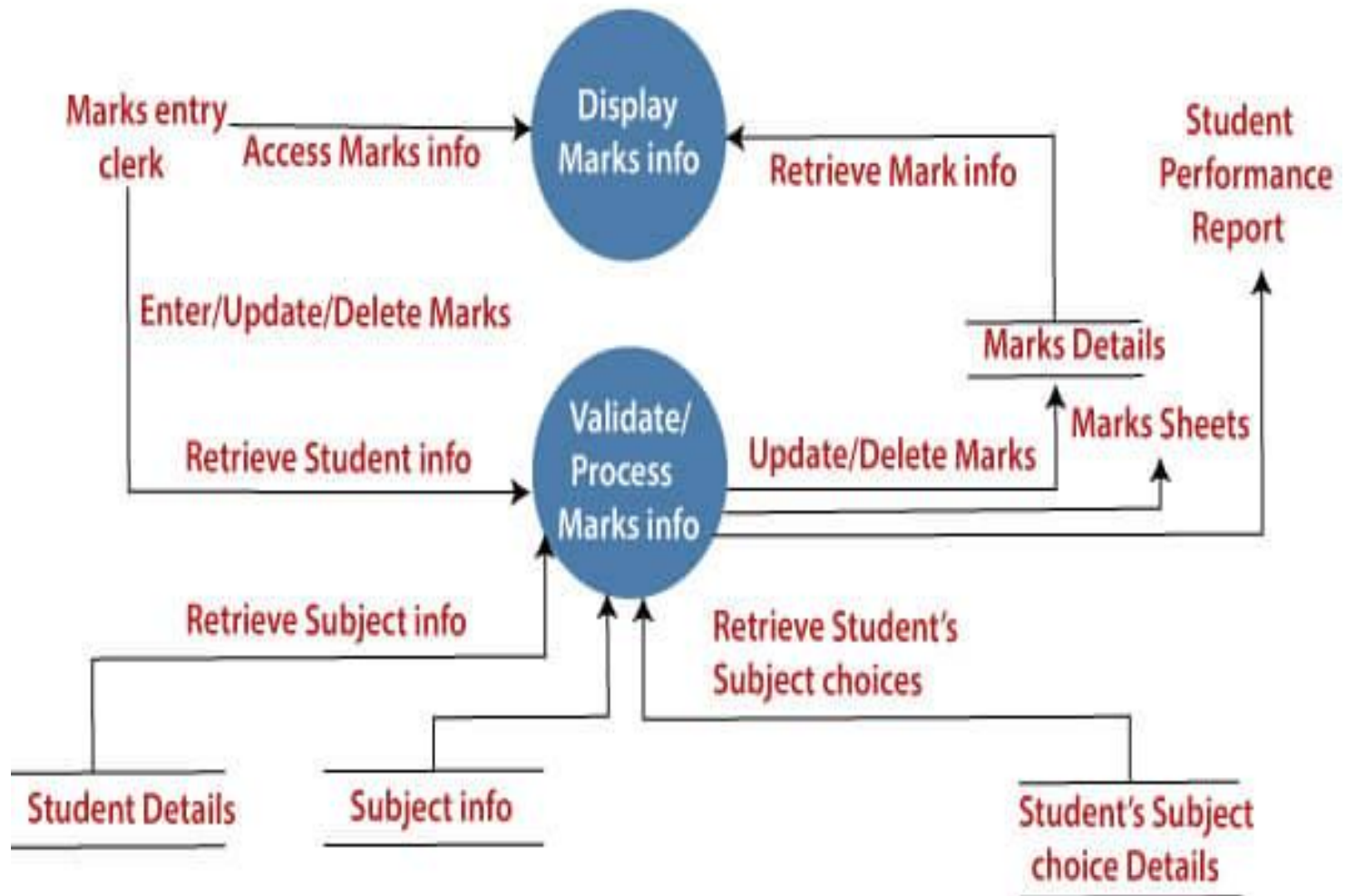
5. Student's Subject Choice Management

The Level 2 DFD of this Process is given below:



6. Marks Information Management

The Level 2 DFD of this Process is given below:



Feasibility Analysis

- **Feasibility Study** in [Software Engineering](#) is a study to evaluate feasibility of proposed project or system.
- Feasibility study is one of stage among important four stages of [Software Project Management Process](#).
- As name suggests feasibility study is the feasibility analysis or it is a measure of the software product in terms of how much beneficial product development will be for the organization in a practical point of view.
- Feasibility study is carried out based on many purposes to analyze whether software product will be right in terms of development, implantation, contribution of project to the organization etc.

Types of Feasibility Study

- The feasibility study mainly concentrates on below five mentioned areas. Among these Economic Feasibility Study is most important part of the feasibility analysis and Legal Feasibility Study is less considered feasibility analysis.

1. **Technical Feasibility:** In Technical Feasibility current resources both hardware software along with required technology are analyzed/assessed to develop project. This technical feasibility study gives report whether there exists correct required resources and technologies which will be used for project development. Along with this, feasibility study also analyzes technical skills and capabilities of technical team, existing technology can be used or not, maintenance and up-gradation is easy or not for chosen technology etc.
2. **Operational Feasibility:** In Operational Feasibility degree of providing service to requirements is analyzed along with how much easy product will be to operate and maintenance after deployment. Along with this other operational scopes are determining usability of product, Determining suggested solution by software development team is acceptable or not etc.

Types of Feasibility Study

3. **Economic Feasibility:** In Economic Feasibility study cost and benefit of the project is analyzed. Means under this feasibility study a detail analysis is carried out what will be cost of the project for development which includes all required cost for final development like hardware and software resource required, design and development cost and operational cost and so on. After that it is analyzed whether project will be beneficial in terms of finance for organization or not.
4. **Legal Feasibility:** In Legal Feasibility study project is analyzed in legality point of view. This includes analyzing barriers of legal implementation of project, data protection acts or social media laws, project certificate, license, copyright etc. Overall it can be said that Legal Feasibility Study is study to know if proposed project conform legal and ethical requirements.
5. **Schedule Feasibility:** In Schedule Feasibility Study mainly timelines/deadlines is analyzed for proposed project which includes how many times teams will take to complete final project which has a great impact on the organization as purpose of project may fail if it can't be completed on time.

Types of Feasibility Study

6. **Cultural and Political Feasibility:** This section assesses how the software project will affect the political environment and organizational culture. This analysis takes into account the organization's culture and how the suggested changes could be received there, as well as any potential political obstacles or internal opposition to the project. It is essential that cultural and political factors be taken into account in order to execute projects successfully.
7. **Market Feasibility:** This refers to evaluating the market's willingness and ability to accept the suggested software system. Analyzing the target market, understanding consumer wants and assessing possible rivals are all part of this study. It assists in identifying whether the project is in line with market expectations and whether there is a feasible market for the good or service being offered.
8. **Resource Feasibility:** This method evaluates if the resources needed to complete the software project successfully are adequate and readily available. Financial, technological and human resources are all taken into account in this study. It guarantees that sufficient hardware, software, trained labor and funding are available to complete the project successfully.

Aim of Feasibility Study

- The overall objective of the organization are covered and contributed by the system or not.
- The implementation of the system be done using current technology or not.
- Can the system be integrated with the other system which are already exist

Feasibility Study Process

The below steps are carried out during entire feasibility analysis.

1. Information assessment: It assesses the original project concept and establishes the main aims and objectives.
2. Information collection: It collects the necessary information and data required to evaluate the project's many components.
3. Report writing: It produces an in-depth feasibility report that details the analysis and results.
4. General information: It gives a summary of the main points discussed in the report on the feasibility study.

Need of Feasibility Study

- Feasibility study is so important stage of [Software Project Management Process](#) as after completion of feasibility study it gives a conclusion of whether to go ahead with proposed project as it is practically feasible or to stop proposed project here as it is not right/feasible to develop or to think/analyze about proposed project again.

Cost/Benefit Analysis

- Cost Benefit analysis is thing that everyone must do so as to think of a powerful or an efficient system. But while thinking out on cost and benefit analysis, we also need to find out factors that really affect benefits and costs of system. In developing cost estimates for a system, we need to consider some of cost elements. Some elements among them are hardware, personnel, facility, operating and supply cost. The following are the cost factors :

1) Hardware cost —
Hardware cost includes actual purchase and peripherals (external devices) that are connected to computer. For example, printer, disk drive etc. Actually, finding actual cost of hardware is generally more difficult especially, when system is shared by various users so as to compared to a system which dedicated stand alone . In some case, best way is to treat it as operating cost.

2) Personnel costs —
Personnel costs includes EDP staff salaries and benefits as well as pay for those who are involved in process of development of system. Cost occurred during development of system which are one time costs and are also called development cost. Once system is installed, cost of operating and maintaining system becomes recurring cost that one has to pay very frequently based on requirement.

3) Facility cost —
Facility cost is amount of money that is spent in preparation of a site that is physical where application or computer will be in operation. This includes wiring, flooring, lighting and air conditioning. These costs are treated as one-time costs and are included into overall cost estimate of candidate system.

4) Operating costs —
These includes all costs associated with day-to-day(everyday) operation of system and amount depends on number of shifts, nature of applications. There are various ways of covering operating costs. One approach is to treat operating costs as an overhead. Another approach is to charge money from each authorized user for amount of processing they require from system. Amount charged is based on computer time or time they spend on system, staff time ad volume of output produced .

5) Supply costs —
Supply cost are variable costs that increase with increased use of paper, disks and like. They should be estimated and included in overall cost of system.

- A system is also expected to provide health benefits. First task is to identify each benefit and then assign some value to it for purpose of cost/benefit analysis. Benefits may be tangible and intangible, direct or indirect.
- Two major benefits are improving performance and minimizing cost of processing of system.
- The performance category emphasizes improvement in accuracy of or access to information and easier access to system by authorized users.
- Minimizing costs through an efficient system – error control or reduction of staff- is a benefit that should be measured and included in cost/benefit analysis.
- The determination of costs and benefit entails following steps :
 1. Identify the costs and benefits pertaining to given project.
 2. Categorize the various costs and benefits for analysis.
 3. Select a method of evaluation.
 4. Interpret the results of the analysis.
 5. Take action

Use Case Diagrams | Unified Modeling Language (UML)

- A Use Case Diagram is a vital tool in system design, it provides a visual representation of how users interact with a system.
- It serves as a blueprint for understanding the functional requirements of a system from a user's perspective, aiding in the communication between stakeholders and guiding the development process.

1. What is a Use Case Diagram in UML?

- A Use Case Diagram is a type of Unified Modeling Language (UML) diagram that represents the interaction between actors (users or external systems) and a system under consideration to accomplish specific goals.
- It provides a high-level view of the system's functionality by illustrating the various ways users can interact with it.

2. Use Case Diagram Notations

- UML notations provide a visual language that enables software developers, designers, and other stakeholders to communicate and document system designs, architectures, and behaviors in a consistent and understandable manner.
- **1.1. Actors**

Actors are external entities that interact with the system. These can include users, other systems, or hardware devices. In the context of a Use Case Diagram, actors initiate use cases and receive the outcomes. Proper identification and understanding of actors are crucial for accurately modeling system behavior.

1.2. Use Cases

- Use cases are like scenes in the play. They represent specific things your system can do. In the online shopping system, examples of use cases could be “Place Order,” “Track Delivery,” or “Update Product Information”. Use cases are represented by ovals.

1.3. System Boundary

- The system boundary is a visual representation of the scope or limits of the system you are modeling. It defines what is inside the system and what is outside. The boundary helps to establish a clear distinction between the elements that are part of the system and those that are external to it. The system boundary is typically represented by a rectangular box that surrounds all the use cases of the system.
- **Purpose of System Boundary:**
- **Scope Definition:** It clearly outlines the boundaries of the system, indicating which components are internal to the system and which are external actors or entities interacting with the system.
- **Focus on Relevance:** By delineating the system's scope, the diagram can focus on illustrating the essential functionalities provided by the system without unnecessary details about external entities.

3. Use Case Diagram Relationships

- In a Use Case Diagram, relationships play a crucial role in depicting the interactions between actors and use cases. These relationships provide a comprehensive view of the system's functionality and its various scenarios. Let's delve into the key types of relationships and explore examples to illustrate their usage.

3.1. Association Relationship

- **Example: Online Banking System**
- **Actor:** Customer
- **Use Case:** Transfer Funds
- **Association:** A line connecting the "Customer" actor to the "Transfer Funds" use case, indicating the customer's involvement in the funds transfer process.

3.2. Include Relationship

- The Include Relationship indicates that a use case includes the functionality of another use case. It is denoted by a dashed arrow pointing from the including use case to the included use case. This relationship promotes modular and reusable design.
- **Example: Social Media Posting**
- **Use Cases:** Compose Post, Add Image
- **Include Relationship:** The “Compose Post” use case includes the functionality of “Add Image.” Therefore, composing a post includes the action of adding an image.
-

3.3. Extend Relationship

- The Extend Relationship illustrates that a use case can be extended by another use case under specific conditions. It is represented by a dashed arrow with the keyword “extend.” This relationship is useful for handling optional or exceptional behavior.
- **Example: Flight Booking System**
- **Use Cases:** Book Flight, Select Seat
- **Extend Relationship:** The “Select Seat” use case may extend the “Book Flight” use case when the user wants to choose a specific seat, but it is an optional step.

3.4. Generalization Relationship

- The Generalization Relationship establishes an “is-a” connection between two use cases, indicating that one use case is a specialized version of another. It is represented by an arrow pointing from the specialized use case to the general use case.
- **Example: Vehicle Rental System**
- **Use Cases:** Rent Car, Rent Bike
- **Generalization Relationship:** Both “Rent Car” and “Rent Bike” are specialized versions of the general use case “Rent Vehicle.”

4. How to draw a Use Case diagram in UML?

- **Step 1: Identify Actors**
- Determine who or what interacts with the system. These are your actors. They can be users, other systems, or external entities.
- **Step 2: Identify Use Cases**
- Identify the main functionalities or actions the system must perform. These are your use cases. Each use case should represent a specific piece of functionality.
- **Step 3: Connect Actors and Use Cases**
- Draw lines (associations) between actors and the use cases they are involved in. This represents the interactions between actors and the system.
- **Step 4: Add System Boundary**
- Draw a box around the actors and use cases to represent the system boundary. This defines the scope of your system.
- **Step 5: Define Relationships**
- If certain use cases are related or if one use case is an extension of another, you can indicate these relationships with appropriate notations.
- **Step 6: Review and Refine**
- Step back and review your diagram. Ensure that it accurately represents the interactions and relationships in your system. Refine as needed.
- **Step 7: Validate**
- Share your use case diagram with stakeholders and gather feedback. Ensure that it aligns with their understanding of the system's functionality.

Let's understand how to draw a Use Case diagram with the help of an Online Shopping System:

- **1. Actors:**

- Customer
- Admin

- **2. Use Cases:**

- Browse Products
- Add to Cart
- Checkout
- Manage Inventory (Admin)

- **3. Relations:**

- The Customer can browse products, add to the cart, and complete the checkout.
- The Admin can manage the inventory.

Requirement Analysis Model/Elements of Requirement Model

- Requirements modelling in software engineering identifies the requirements that a software application or system must meet in order to solve the business problem. Requirements are divided into functional (what the system will have to do) and non-functional (constraints within which the system will have to perform).
- Their are common types that make up a requirement model are Scenario-based model, Class-based model, Behavioural model and Flow oriented Model.
- The different modeling technique is individually understandable. However, it is highly important to know how to use these models and when to use them.

Requirement Analysis

- Requirement Analysis is significant and essential activity after elicitation.
- This activity reviews all requirements and may provide a graphical view of the entire system.
- After the completion of the analysis the understandability of the project may improve significantly

Requirement Analysis Model

1) Scenario based Modelling/Element

- User Stories
- Use Case Diagram
- Activity Diagram

2)Class Based Modelling/Element

- Class Diagram
- Collaboration Diagram

3) Behavioural Based Modelling/Element

- State Transition Diagram
- Sequence Diagram

4) Flow Based Modelling/Element

- Data Flow Diagram
- Control Flow Diagram

1) Scenario based Modelling/Element

1)Use Case Diagram

- A Use Case Diagram is a vital tool in system design, it provides a visual representation of how users interact with a system.
- It serves as a blueprint for understanding the functional requirements of a system from a user's perspective, aiding in the communication between stakeholders and guiding the development process.

1) Scenario based Modelling/Element

2) Activity Diagram

- Activity diagrams show the flow of one activity to another within a system or process.
- Even complex systems can be visualized using activity diagrams.
- They are used within organizations to model customer journeys, to show the process of receiving an order through shipping to the customer, and to model sales processes.
- It is a type of [behavioral diagram](#).

2) Activity Diagram

- An activity diagram is a flowchart of activities, as it represents the workflow among various activities.
- They are identical to the flowcharts, but they themselves are not exactly the flowchart.
- In other words, it can be said that an activity diagram is an enhancement of the flowchart, which encompasses several unique skills.
- Following are the rules that are to be followed for drawing an activity diagram:
 1. A meaningful name should be given to each and every activity.
 2. Identify all of the constraints.
 3. Acknowledge the activity associations.

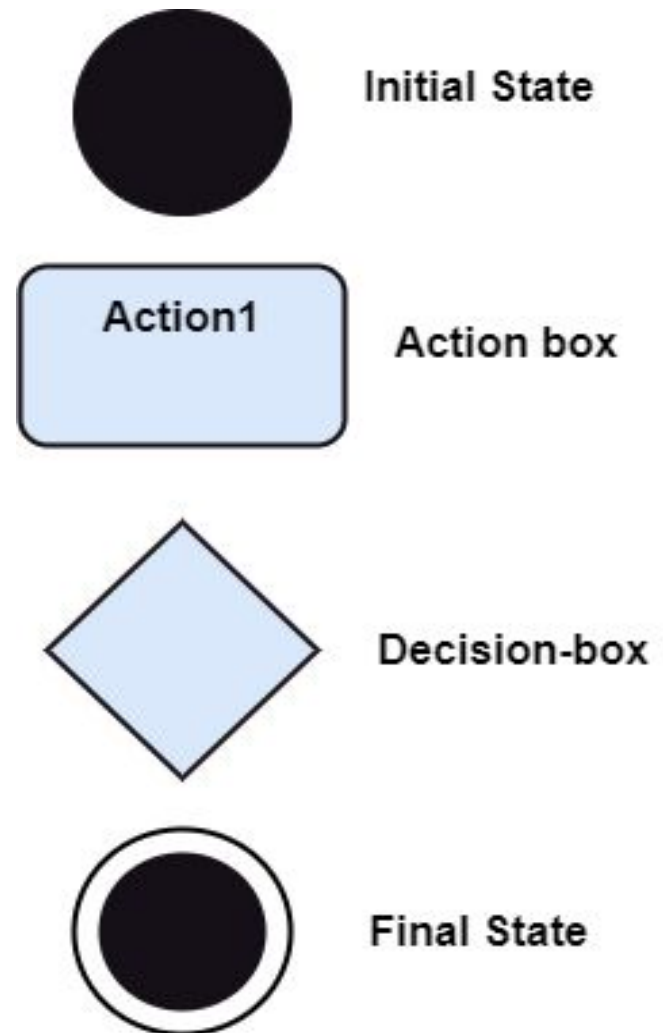
Notation of an Activity diagram

Initial State: It depicts the initial stage or beginning of the set of actions.

Final State: It is the stage where all the control flows and object flows end.

Decision Box: It makes sure that the control flow or object flow will follow only one path.

Action Box: It represents the set of actions that are to be performed.



When to use an Activity Diagram?

- An activity diagram can be used to portray business processes and workflows. Also, it used for modeling business as well as the software. An activity diagram is utilized for the followings:
- To graphically model the workflow in an easier and understandable way.
- To model the execution flow among several activities.
- To model comprehensive information of a function or an algorithm employed within the system.
- To model the business process and its workflow.
- To envision the dynamic aspect of a system.
- To generate the top-level flowcharts for representing the workflow of an application.
- To represent a high-level view of a distributed or an object-oriented system.

2)Class Based Modelling/Element

1) Class Diagram

- Class diagram is a static diagram.
- It represents the static view of an application.
- Class diagram is not only used for visualizing, describing, and documenting different aspects of a system but also for constructing executable code of the software application.
- Class diagram describes the attributes and operations of a class and also the constraints imposed on the system.
- The class diagrams are widely used in the modeling of objectoriented systems because they are the only UML diagrams, which can be mapped directly with object-oriented languages.

1) Class Diagram

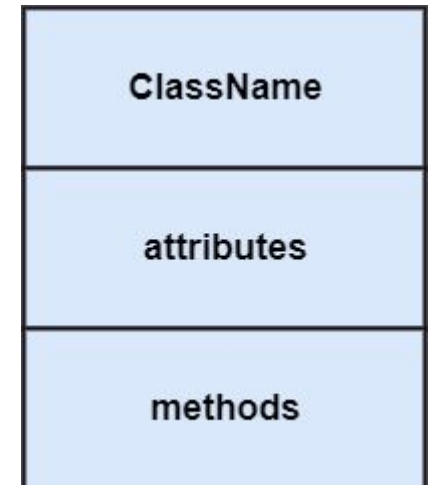
- Class diagram shows a collection of classes, interfaces, associations, collaborations, and constraints. It is also known as a structural diagram.

The purpose of the class diagram can be summarized as –

- Analysis and design of the static view of an application.
- Describe responsibilities of a system.
- Base for component and deployment diagrams.
- Forward and reverse engineering.

The class diagram is made up of three sections:

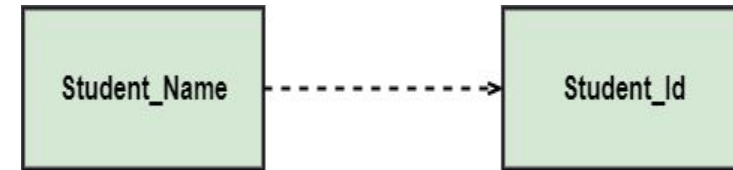
- **Upper Section:** The upper section encompasses the name of the class. A class is a representation of similar objects that shares the same relationships, attributes, operations, and semantics. Some of the following rules that should be taken into account while representing a class are given below:
 - Capitalize the initial letter of the class name.
 - Place the class name in the center of the upper section.
 - A class name must be written in bold format.
 - The name of the abstract class should be written in italics format.
- **Middle Section:** The middle section constitutes the attributes, which describe the quality of the class.
- **Lower Section:** The lower section contain methods or operations. The methods are represented in the form of a list, where each method is written in a single line. It demonstrates how a class interacts with data.



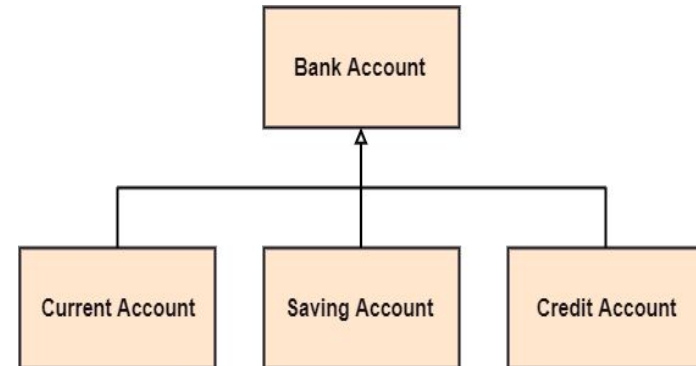
Relationships

- In UML, relationships are of three types:

1) Dependency: A dependency is a semantic relationship between two or more classes where a change in one class cause changes in another class. It forms a weaker relationship. In the following example, Student_Name is dependent on the Student_Id.



2) Generalization: A generalization is a relationship between a parent class (superclass) and a child class (subclass). In this, the child class is inherited from the parent class. For example, The Current Account, Saving Account, and Credit Account are the generalized form of Bank Account.



3) Association: It describes a static or physical connection between two or more objects. It depicts how many objects are there in the relationship. For example, a department is associated with the college.



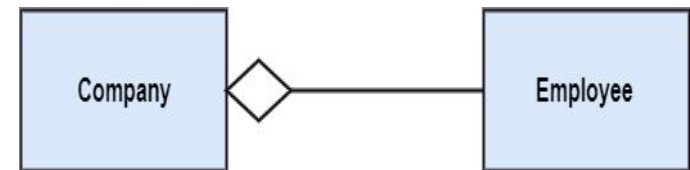
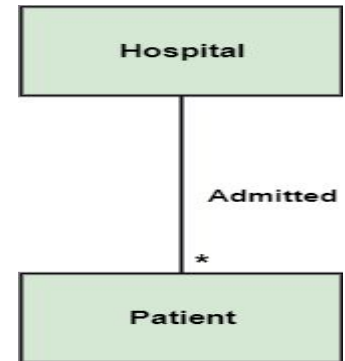
- **Multiplicity:** It defines a specific range of allowable instances of attributes. In case if a range is not specified, one is considered as a default multiplicity.

For example, multiple patients are admitted to one hospital.

- **Aggregation:** An aggregation is a subset of association, which represents has a relationship. It is more specific than association. It defines a part-whole or part-of relationship. In this kind of relationship, the child class can exist independently of its parent class.

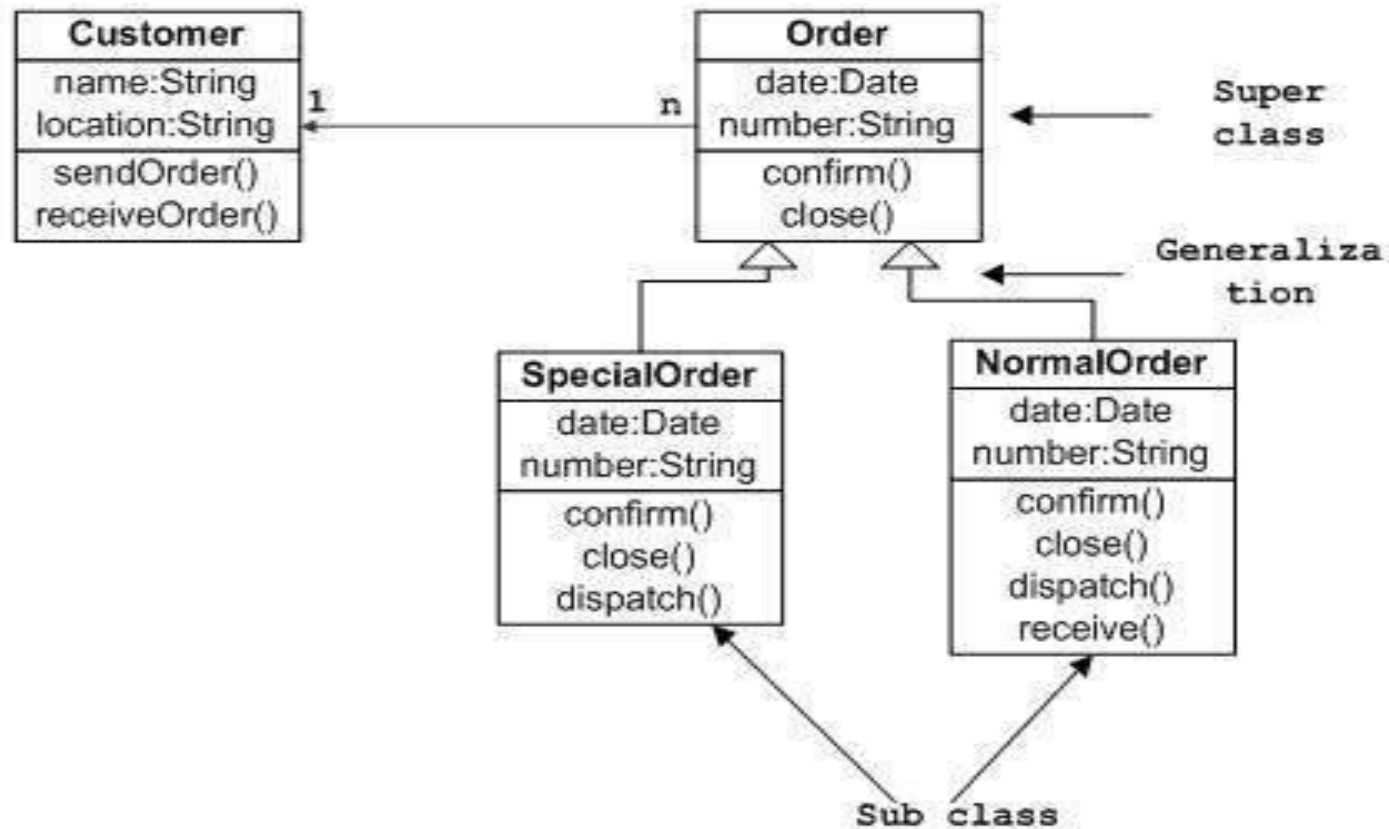
The company encompasses a number of employees, and even if one employee resigns, the company still exists

- **Composition:** The composition is a subset of aggregation. It portrays the dependency between the parent and its child, which means if one part is deleted, then the other part also gets discarded. It represents a whole-part relationship.
- A contact book consists of multiple contacts, and if you delete the contact book, all the contacts will be lost.



1) Class Diagram

Sample Class Diagram



2) Collaboration Diagram

- The collaboration diagram is used to show the relationship between the objects in a system.
- Both the sequence and the collaboration diagrams represent the same information but differently.
- Instead of showing the flow of messages, it depicts the architecture of the object residing in the system as it is based on object-oriented programming.
- An object consists of several features. Multiple objects present in the system are connected to each other.
- The collaboration diagram, which is also known as a communication diagram, is used to portray the object's architecture in the system.

Notations of a Collaboration Diagram

- Following are the components of a componen

1. **Objects:** The representation of an object is done by an object symbol with its name and class underlined, separated by a colon.

In the collaboration diagram, objects are utilized in the following ways:

1. The object is represented by specifying their name and class.
2. It is not mandatory for every class to appear.
3. A class may constitute more than one object.
4. In the collaboration diagram, firstly, the object is created, and then its class is specified.
5. To differentiate one object from another object, it is necessary to name them.

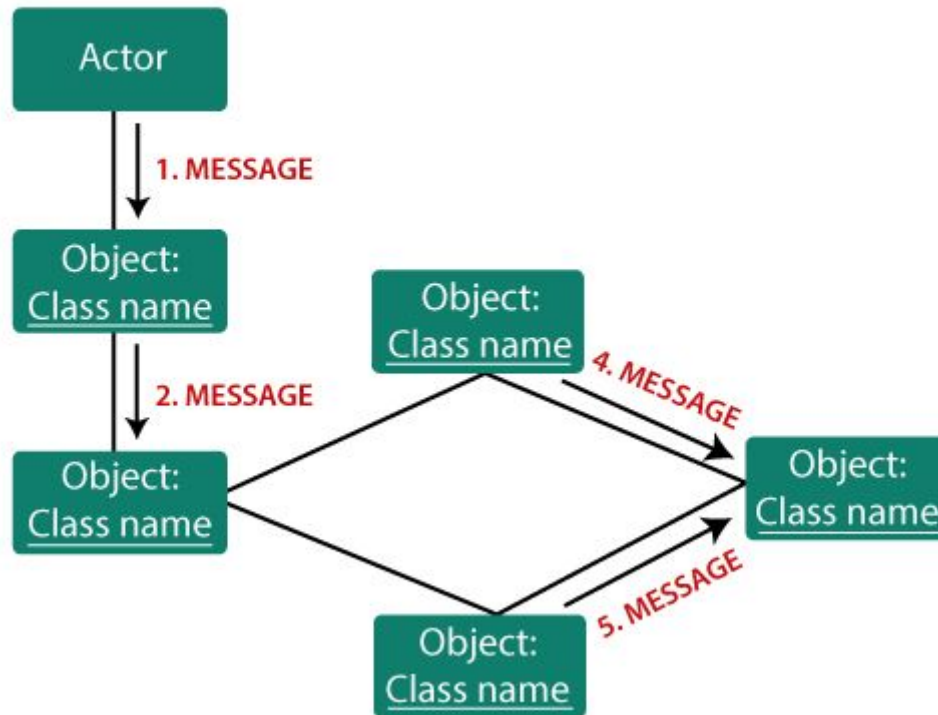
2. **Actors:** In the collaboration diagram, the actor plays the main role as it invokes the interaction. Each actor has its respective role and name. In this, one actor initiates the use case.

3. **Links:** The link is an instance of association, which associates the objects and actors. It portrays a relationship between the objects through which the messages are sent. It is represented by a solid line. The link helps an object to connect with or navigate to another object, such that the message flows are attached to links.

4. **Messages:** It is a communication between objects which carries information and includes a sequence number, so that the activity may take place. It is represented by a labeled arrow, which is placed near a link. The messages are sent from the sender to the receiver, and the direction must be navigable in that particular direction. The receiver must understand the message.

2) Collaboration Diagram

Components of a collaboration diagram



3) Behavioural Based Modelling/Element

- State Transition Diagram
- Sequence Diagram

1)State Transition Diagram

- **State Transition Diagram** are also known as Dynamic models.
- As the name suggests, it is a type of diagram that is used to represent different transition (changing) states of a System.
- It is generally used to graphically represent all possible transition states a system can have and model such systems.
- It is very essential and important and right for object-oriented modeling from the beginning.
- The System consists of various states that are being represented using various symbols in the state transition diagram.

**You can see the symbols and their description
given below :**

• **Initial State –**



• **Final State –**



• **Simple State –**



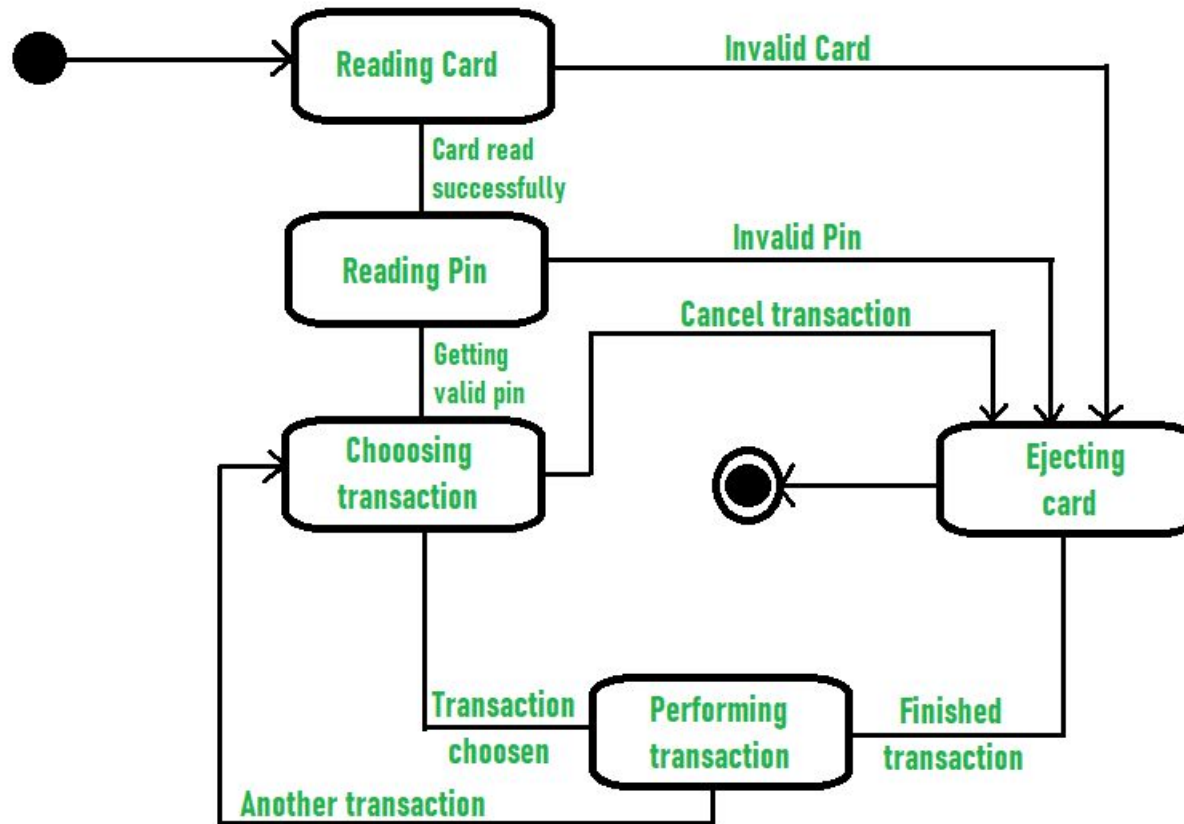
• **Composite State –**



1)State Transition Diagram

Type of State	Description
Initial State	In a System, it represents Starting state.
Final State	In a System, it represents Ending state.
Simple State	In a System, it represents a Simple state with no substructure.
Composite State	In a System, it represents a Composite state with two or more parallel or concurrent states out of which only one state will be active at a time and other states will be inactive.

Now let us see the State Transition Diagram of Automated Teller Machine (ATM) System. In this you will see the processing when the customer performs transactions using ATM card.



State Transition Diagram for ATM System

Automated Teller Machine (ATM) System

- When the customer inserts the bank or credit card in the ATM's card reader, the entry action i.e readcard is performed by the ATM machine.
- If the card is not valid then the machine will perform exit action.
- After the card is being read successfully, the ATM machine will ask for Pin.
- Then the customer enters the pin and ATM machine then reads pin. If the pin entered is not valid then machine will perform exit action.
- If the pin entered is valid, then the machine further process towards transaction.
- After successful transaction, machine undergoes the exit action i.e., ejectcard that discharges the customer's card.

2)Sequence Diagram

- Unified Modelling Language (UML) is a modeling language in the field of software engineering that aims to set standard ways to visualize the design of a system.
- UML guides the creation of multiple types of diagrams such as interaction, structure, and behavior diagrams.
- A **sequence diagram** is the most commonly used **interaction** diagram

2)Sequence Diagram

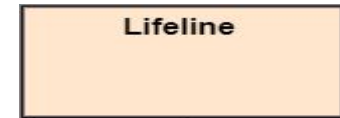
Purpose of a Sequence Diagram

- To model high-level interaction among active objects within a system.
- To model interaction among objects inside a collaboration realizing a use case.
- It either models generic interactions or some certain instances of interaction.

1. Sequence Diagram Notation

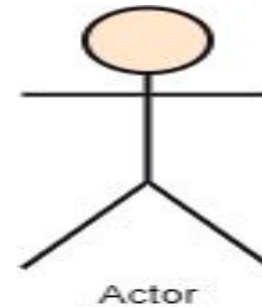
1. Lifeline

- An individual participant in the sequence diagram is represented by a lifeline. It is positioned at the top of the diagram.



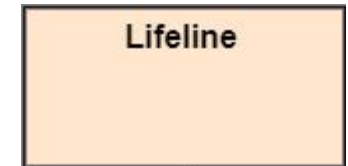
2. Actors

- An actor in a UML diagram represents a type of role where it interacts with the system and its objects. It is important to note here that an actor is always outside the scope of the system we aim to model using the UML diagram.



3. Activation

- It is represented by a thin rectangle on the lifeline. It describes that time period in which an operation is performed by an element, such that the top and the bottom of the rectangle is associated with the initiation and the completion time, each respectively.

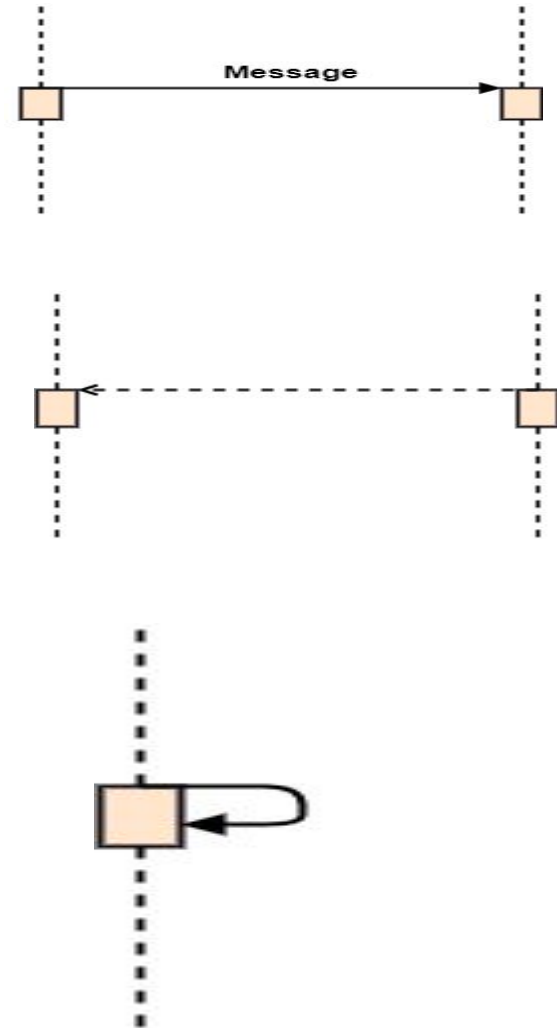


Messages

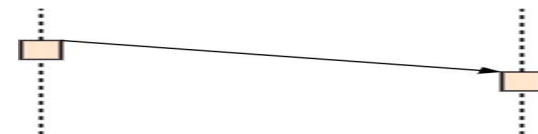
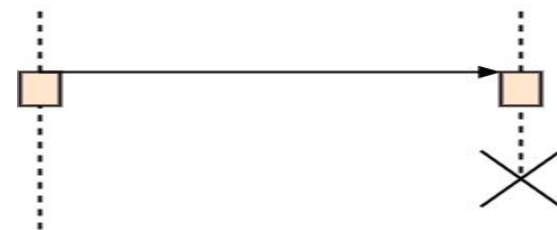
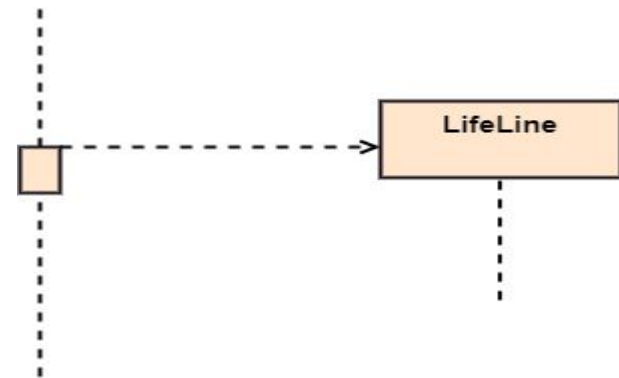
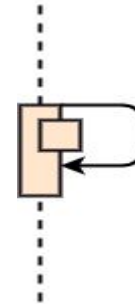
The messages depict the interaction between the objects and are represented by arrows. They are in the sequential order on the lifeline. The core of the sequence diagram is formed by messages and lifelines.

Following are types of messages enlisted below:

- **Call Message:** It defines a particular communication between the lifelines of an interaction, which represents that the target lifeline has invoked an operation.
- **Return Message:** It defines a particular communication between the lifelines of interaction that represent the flow of information from the receiver of the corresponding caller message.
- **Self Message:** It describes a communication, particularly between the lifelines of an interaction that represents a message of the same lifeline, has been invoked.



- **Recursive Message:** A self message sent for recursive purpose is called a recursive message. In other words, it can be said that the recursive message is a special case of the self message as it represents the recursive calls.
- **Create Message:** It describes a communication, particularly between the lifelines of an interaction describing that the target (lifeline) has been instantiated.
- **Destroy Message:** It describes a communication, particularly between the lifelines of an interaction that depicts a request to destroy the lifecycle of the target.
- **Duration Message:** It describes a communication particularly between the lifelines of an interaction, which portrays the time passage of the message while modeling a system.



4) Flow Based Modelling/Element

- Data Flow Diagram
- Control Flow Diagram

1)Data Flow Diagram

- A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. A neat and clear DFD can depict the right amount of the system requirement graphically. It can be manual, automated, or a combination of both.
- It shows how data enters and leaves the system, what changes the information, and where data is stored.
- The objective of a DFD is to show the scope and boundaries of a system as a whole. It may be used as a communication tool between a system analyst and any person who plays a part in the order that acts as a starting point for redesigning a system. The DFD is also called as a data flow graph or bubble chart.

2) Control Flow Diagram

- A **Control Flow Graph (CFG)** is the graphical representation of control flow or [computation during the execution of programs](#) or applications.
- Control flow graphs are mostly used in static analysis as well as compiler applications, as they can accurately represent the flow inside a program unit.
- The control flow graph was originally developed by *Frances E. Allen*..

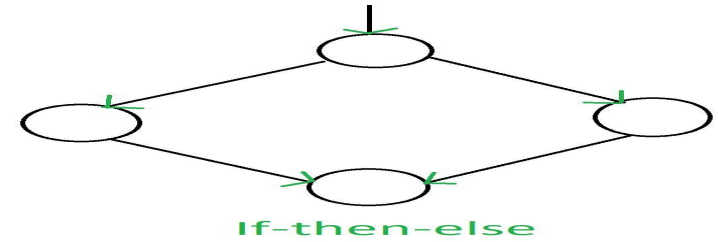
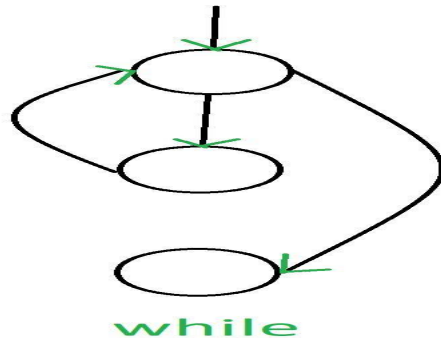
Characteristics of Control Flow Graph

- The control flow graph is process-oriented.
- The control flow graph shows all the paths that can be traversed during a program execution.
- A control flow graph is a directed graph.
- Edges in CFG portray control flow paths and the nodes in CFG portray basic blocks.

There exist 2 designated blocks in the Control Flow Graph:

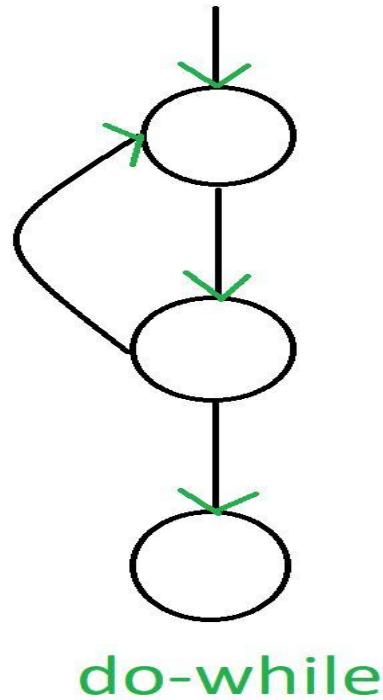
- **Entry Block:** The entry block allows the control to enter into the control flow graph.
- **Exit Block:** Control flow leaves through the exit block.

- 1. If-else

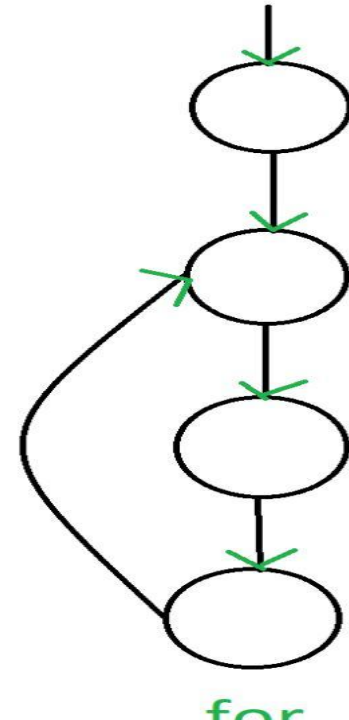


- 2. While

- 3. do-while

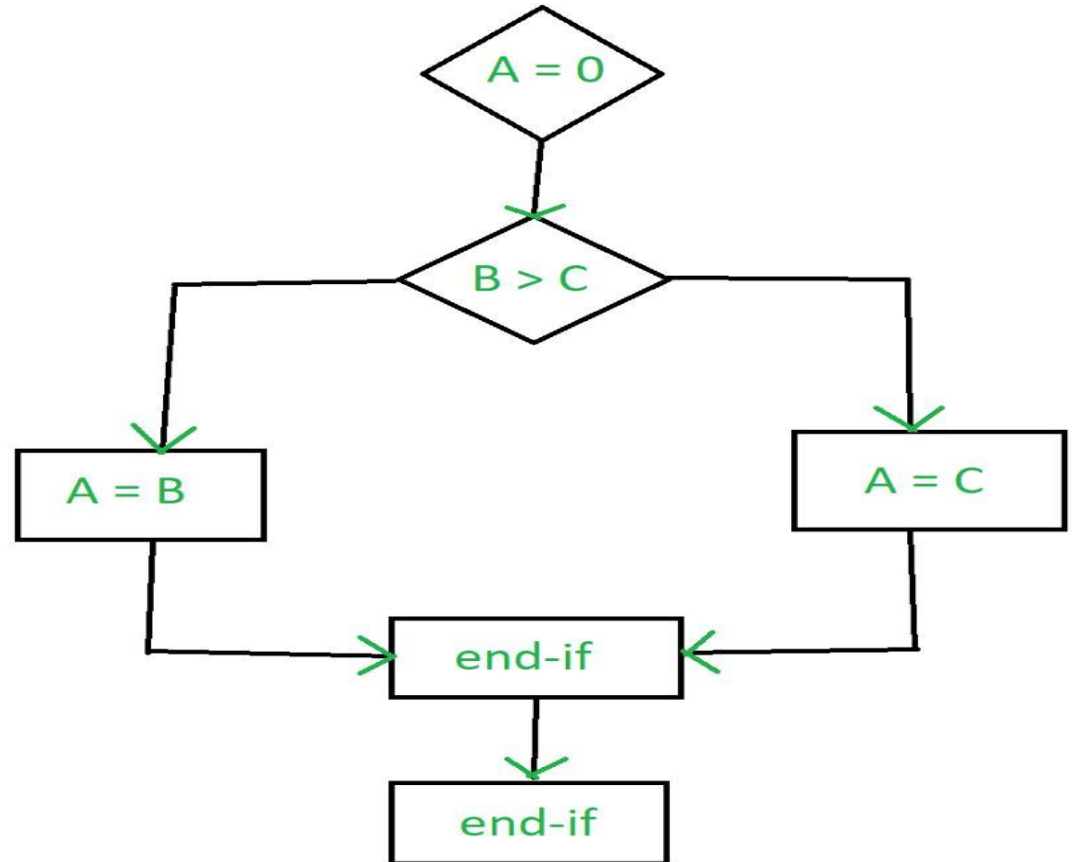


- 4. for

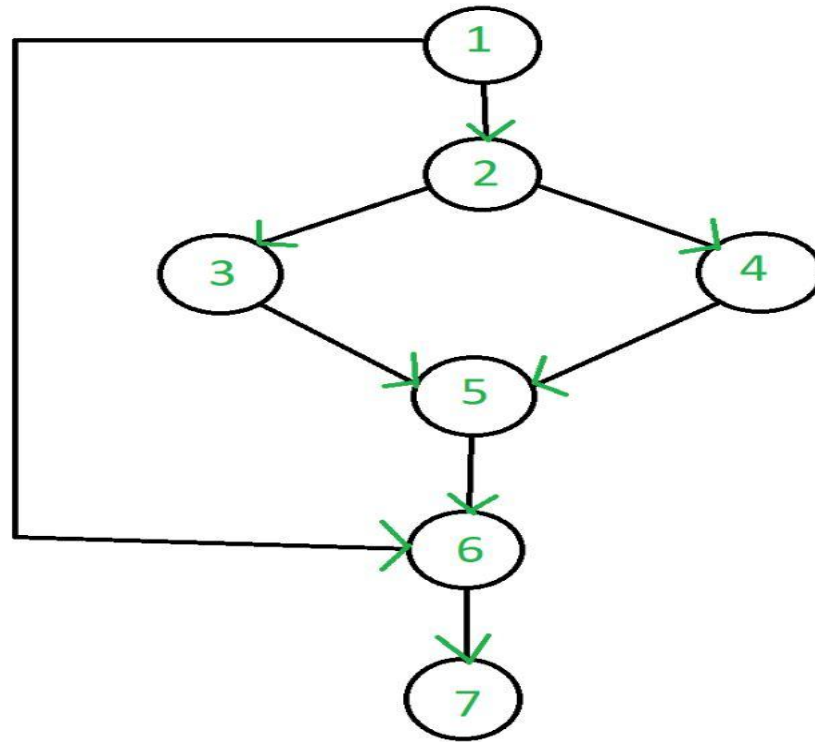


Example

- *if* $A = 10$ *then*
- *if* $B > C$
- $A = B$
- *else* $A = C$
- *endif*
- *print* A, B, C
- Flowchart of above example will be:



- Control Flow Graph of above example will be:



Control Flow Graph

Advantage of CFG

- There are many advantages of a control flow graph.
- It can easily encapsulate the information per each basic block.
- It can easily locate inaccessible codes of a program and syntactic structures such as loops are easy to find in a control flow graph.