Software Requirements Analysis and Modeling

- ❖ **Requirement Engineering**
- ❖ **Requirement Modeling**
- ❖ **Data Flow Diagram (DFD)**
- ❖ **Scenario Based Model**
- ❖ **Software Requirement Specification document format (IEEE)**

Requirement Engineering

❖ What is Requirement Engineering?

- Requirement Engineering is the process of defining, documenting and maintaining the requirements.
- It is a process of gathering and defining service provided by the system.
- The process of establishing the services that the customer requires from a system and the constraints under which it operates and is developed.
- Requirements Engineering Process consists of the following main activities:
  - Feasibility Study
  - Requirement Elicitation and Analysis
  - Software Requirement Specification
  - Software Requirement Verification and Validation
  - Software Requirement Management

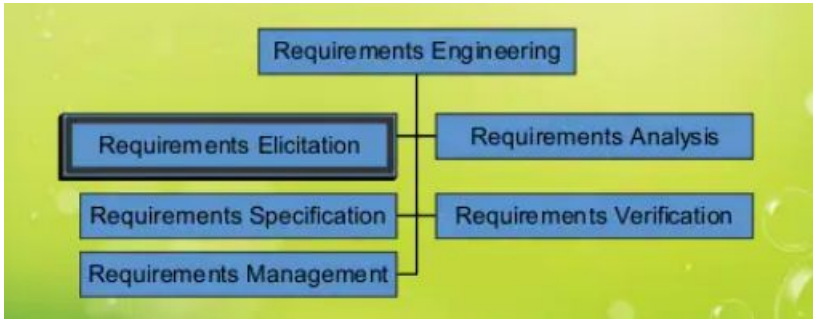
Requirement Engineering …. Contd.

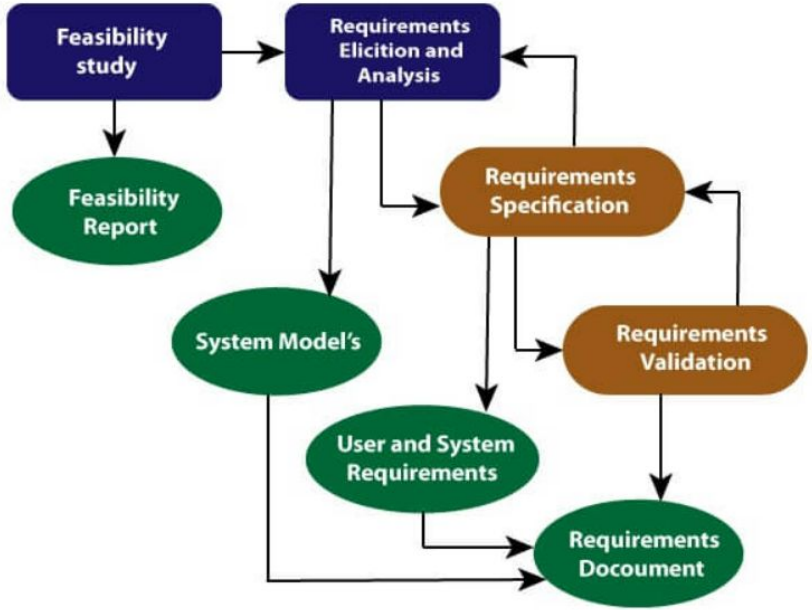
❖ Importance of Requirement

- A requirements mandates that something be accomplished, transformed, produced or provided.
  - Making design decisions without understanding all the constraints on the system to be developed results in a system which fails to meet customer's expectations.
  - Cost of correcting errors increases as the design process advances.
  - An error detected in the maintenance phase is **20** times as costly to fix as an error detected in the coding stage.

Requirement Engineering Process

- The processes used for Requirement Engineering vary widely depending on the application domain, the people involved and the organization developing the requirements.





Requirement Engineering Process ….Contd.

 Requirements Elicitation:

- The process through which clients and developers review, articulate and understand the needs of the clients and the constraints on the software.
- It requires involvement with the client, domain experts, and end-users in order to establish the client's needs and constraints on the system.
- Following are some of the techniques used for interaction:
  - ✔ Interviews
  - ✔ Questionaries
  - ✔ Focus groups
  - ✔ Apprenticing
  - ✔ Modelling

Requirement Engineering Process ….Contd.

 Requirements Analysis:

- The process of analyzing the needs of the clients in order to arrive at a definition of the requirements.
- It aims to deepen our understanding of the constraints and the client's needs.

Requirements Specification:

- The process by which a document is developed which clearly communicates the requirements.
- The requirements are captured, or expressed, or articulated, in a software requirements specification.

Requirement Engineering Process ….Contd.

 Requirements Verification and Validation:

- The process of ensuring that the requirements and the Software Requirements Specification are in compliance with the needs of the clients and the system.
- Techniques included are:
  - ✔ Reviews, inspections and walkthroughs of requirements
  - ✔ Prototyping

Requirements Management and Evolution:

- It involves the planning and controlling of the requirements engineering processes.
- Requirements specification should evolve with time.

Requirement Engineering Process ….Contd.

- Feasibility
  - Feasibility Studies aim to objectively and rationally
    - Uncover the strengths and weaknesses of the existing business or proposed venture.
    - Opportunities and threats as presented by the environment.
    - The resources required to carry through.
    - Ultimately the prospects for success of the proposition.
  - The two criteria to judge feasibility are cost required and value to be attained.
  - The assessment is based on an outline design of system requirements in terms of Input, Processes, Output, Fields, Programs, and Procedures.

Feasibility

 Types of Feasibility

I. Technical Feasibility:
- It is carried out to determine whether the company has the capability, in terms of software, hardware, personnel and expertise, to handle the completion of the project.
- It evaluates the current technologies, which are needed to accomplish customer requirements within the time and budget.

II. Operational Feasibility:
- The 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.

Feasibility …. Contd.

III. Economic Feasibility:

- It decides whether the necessary software can generate financial profits for an organization.

IV. Legal Feasibility:

- The 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.

V. Schedule Feasibility:

- Mainly timelines/deadlines are analyzed for proposed project which includes how much time teams will take to complete final project.
- This has a great impact on the organization as purpose of project may fail if it cannot be completed on time.

Requirements Elicitation

- It is the process of discovering the requirements for a system by communication with the customers, system users, and others who have a stake in the system development.
- The success of an elicitation technique used depends on the maturity of the analyst, developers, users, and the customer involved.
- There are a number of requirements elicitation methods:-

a) Interviews:

- Objective of conducting an interview is to understand the customer's expectations from the software.
- It is impossible to interview every stakeholder hence representatives from groups are selected based on their expertise and credibility.
- Interviews are strong medium to collect requirements.
- Organization may conduct several types of interviews such as:

Requirements Elicitation …Contd.

- Structured (closed) interviews:
  - » In structured interview, agenda of fairly open questions is prepared.
  - » Sometimes a proper questionnaire is designed for the interview.
  - » Every single information to gather is decided in advance, they follow pattern and matter of discussion firmly.
- Non-structured (open) interviews:
  - » In open-ended interviews there is no pre-set agenda.
  - » Context free questions may be asked to understand the problem.
  - » The information to gather is not decided in advance, more flexible and less biased.
- Oral interviews
- Written interviews
- One-to-one interviews which are held between two persons across the table.
- Group interviews which are held between groups of participants.
  - » They help to uncover any missing requirement as numerous people are involved.

Requirements Elicitation …Contd.

b) Surveys:

- Organization may conduct surveys among various stakeholders by querying about their expectation and requirements from the upcoming system.

c) Questionnaires:

- A document with pre-defined set of objective questions and respective options is handed over to all stakeholders to answer, which are collected and compiled.
- A shortcoming of this technique is, if an option for some issue is not mentioned in the questionnaire, the issue might be left unattended.

d) Task analysis:

- Team of engineers and developers may analyze the operation for which the new system is required.
- If the client already has some software to perform certain operation, it is studied and requirements of proposed system are collected.

Requirements Elicitation …Contd.

e) Domain Analysis:

- Every software falls into some domain category.
- The expert people in the domain can be a great help to analyze general and specific requirements.

f) Brainstorming:

- An informal debate is held among various stakeholders and all their inputs are recorded for further requirements analysis.
- It is a group technique.
- It is intended to generate lots of new ideas hence providing a platform to share views.
- A highly trained facilitator is required to handle group bias and group conflicts.
- Every idea is documented so that everyone can see it.
- Finally, a document is prepared which consists of the list of requirements and their priority if possible.

Requirements Elicitation …Contd.

g) Prototyping:

- Prototyping is building user interface without adding detail functionality for user to interpret the features of intended software product.
- It helps giving better idea of requirements.
- If there is no software installed at client's end for developer's reference and the client is not aware of its own requirements, the developer creates a prototype based on initially mentioned requirements.
- The prototype is shown to the client and the feedback is noted.
- The client feedback serves as an input for requirement gathering.

h) Observation:

- Team of experts visit the client's organization or workplace.
- They observe the actual working of the existing installed systems. They observe the workflow at client's end and how execution problems are dealt.
- The team itself draws some conclusions which aid to form requirements expected from the software.

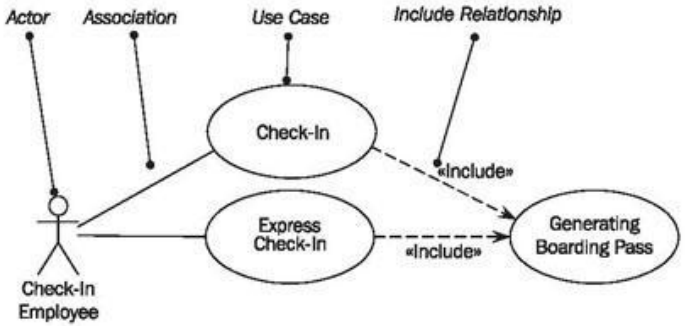
Requirements Elicitation …Contd.

i) **Joint Application Development (JAD):**
- It is a methodology that involves the client or end user in the design and development of an application, through a succession of collaborative workshops called JAD sessions.
- It leads to faster development times and greater client satisfaction, because the client is involved throughout the development process.
- The developer investigates the system requirements and develops an application, with client input consisting of a series of interviews.

j) **Use Case Approach:**
- This technique combines text and pictures to provide a better understanding of the requirements.
- The use cases describe the ‘what’, of a system and not ‘how’. Hence, they only give a functional view of the system.
- The components of the use case design includes few major things – Actor, Use cases, relationship, use case diagram.

Requirements Elicitation …Contd.

- ***Actor***
  - –It is the external agent that lies outside the system but interacts with it in some way.
  - –An actor maybe a person, machine etc.
  - –It is represented as a stick figure.
  - –Actors can be primary actors or secondary actors.
    - » Primary actors – It requires assistance from the system to achieve a goal.
    - » Secondary actor – It is an actor from which the system needs assistance.



Requirements Elicitation …Contd.

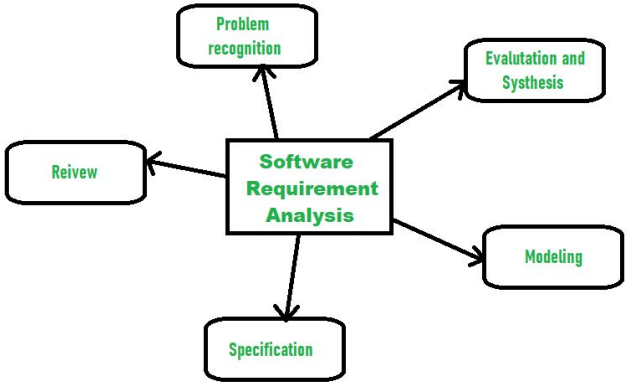
- ***Use cases***
  - They describe the sequence of interactions between actors and the system.
  - They capture who(actors) do what(interaction) with the system.
  - A complete set of use cases specifies all possible ways to use the system.
  - An oval is used to represent a use case.
- ***Relationship***
  - A line is used to represent a relationship between an actor and a use case.
- ***Use case diagram***
  - A use case diagram graphically represents what happens when an actor interacts with a system.
  - It captures the functional aspect of the system.

Requirements Analysis

- It is the process of studying and analysing the customer and the user/ stakeholder needs to arrive at a definition of software requirements.
- It is a significant and essential activity after elicitation.
- Here, analysis, refinement, and scrutinizing of the gathered requirements is done to make consistent and unambiguous requirements.
- This activity reviews all requirements and may provide a graphical view of the entire system.
- After the completion of the analysis, it is expected that the understandability of the project may improve significantly.
- Customer interaction can also be done to clarify points of confusion and to understand which requirements are more important than others.
- Requirements Analysis is important due to the following reasons:
  - ✔ Customer may provide some requirements which could be impossible to implement due to various dependencies.
  - ✔ A business analyst might have understood the requirement from the customer differently than how a programmer would have interpreted.

Requirements Analysis …. Contd.

There are several activities involved in analyzing Software requirements:



I.Problem Recognition:

- The main aim of requirement analysis is to fully understand main objective of requirement that includes why it is needed, does it add value to product, will it be beneficial, does it increase quality of the project, does it will have any other effect.
- All these points are fully recognized in problem recognition so that requirements that are essential can be fulfilled to solve business problems.

Requirements Analysis …. Contd.

II. **Evaluation and Synthesis**:

- Evaluation means judgment about something whether it is worth or not and synthesis means to create or form something.
- Some tasks that are important in the evaluation and synthesis of software requirement :
  - To define all functions of software that necessary.
  - To define all data objects that are present externally and are easily observable.
  - To evaluate that flow of data is worth or not.
  - To fully understand overall behavior of system that means overall working of system.
  - To identify and discover constraints that are designed.
  - To define and establish character of system interface to fully understand how system interacts with two or more components or with one another.

Requirements Analysis …. Contd.

III. **Modeling** :

- After complete gathering of information from above tasks, functional and behavioral models are established after checking function and behavior of system using a domain model that also known as the conceptual model.

IV. **Specification**:

- The software requirement specification (SRS) which means to specify the requirement whether it is functional or non-functional should be developed.

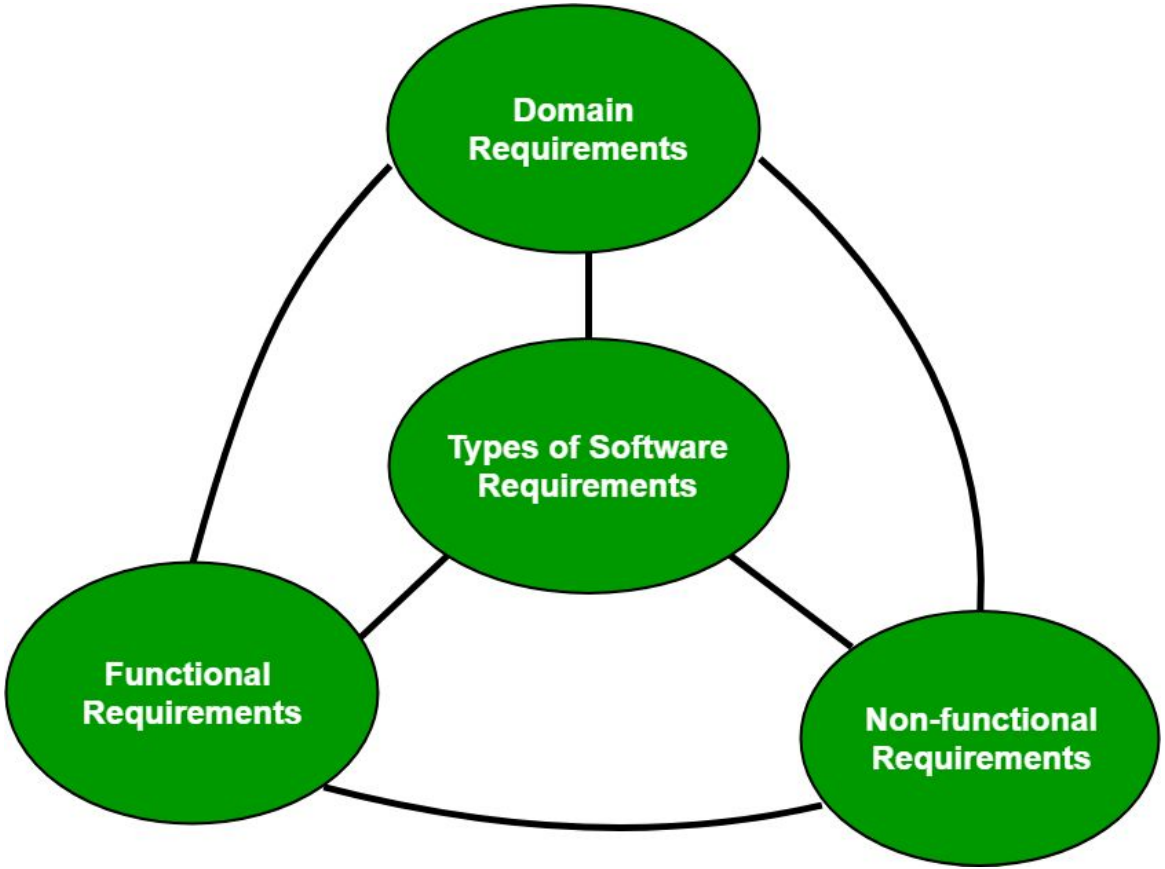
V. **Review**:

- After developing the SRS, it must be reviewed to check whether it can be improved or not and must be refined to make it better and increase the quality.

Requirements Specification

- Requirement Perspectives:
  a) User Requirement
     - Statements in natural language plus diagrams of the services the system provides and its operational constraints.
     - It is written for customers.
  b) System Requirement
     a) A structured document setting out detailed descriptions of the system's functions, services and operational constraints.
     b) It defines what should be implemented so as to be part of a contract between the client and the contractor.
- Types of Requirements:
  i. Functional requirements
  ii. Non-functional requirements
  iii. Domain requirements

Requirements Specification …. Contd.



Requirements Specification …. Contd.

- **Functional Requirements:**
  - These are the requirements that the end user specifically demands as basic facilities that the system should offer.
  - All these functionalities need to be necessarily incorporated into the system as a part of the contract.
  - These are represented or stated in the form of input to be given to the system, the operation performed and the output expected.
  - They are basically the requirements stated by the user which one can see directly in the final product, unlike the non-functional requirements.
  - There are many ways of expressing functional requirements e.g., natural language, a structured or formatted language with no rigorous syntax and formal specification language with proper syntax.

Requirements Specification …. Contd.

- **Non-Functional Requirements:**
  - These are basically the quality constraints that the system must satisfy according to the project contract.
  - The priority or extent to which these factors are implemented varies from one project to other.
  - They are also called non-behavioral requirements.

❖ NFR's are classified into following types:

- Interface constraints
- Performance constraints: response time, security, storage space, etc.
- Operating constraints
- Life cycle constraints: maintainability, portability, etc.
- Economic constraints

Requirements Specification …. Contd.

- The process of specifying non-functional requirements requires the knowledge of the functionality of the system, as well as the knowledge of the context within which the system will operate.

- They basically deal with issues like:
  - ✔ Portability
  - ✔ Security
  - ✔ Maintainability
  - ✔ Reliability
  - ✔ Scalability
  - ✔ Performance
  - ✔ Reusability
  - ✔ Flexibility

Requirements Specification …. Contd.

- **Domain requirements**:
  - Domain requirements are the requirements which are characteristic of a particular category or domain of projects.
  - The basic functions that a system of a specific domain must necessarily exhibit come under this category.
  - For instance, in an academic software that maintains records of a school or college, the functionality of being able to access the list of faculty and list of students of each grade is a domain requirement.
  - These requirements are therefore identified from that domain model and are not user specific.

Requirements Characteristics

- A software requirements specification (SRS) is a document that describes what the software will do and how it will be expected to perform.
- It also describes the functionality the product needs to fulfill all stakeholders (business, users) needs.
- Following are the characteristics of a good SRS document:

i. *Correctness*:
   - User review is used to ensure the correctness of requirements stated in the SRS.
   - SRS is said to be correct if it covers all the requirements that are actually expected from the system.

ii. *Completeness*:
   - Completeness of SRS indicates every sense of completion including the numbering of all the pages, covering all the functional and non-functional requirements properly.

Requirements Characteristics ..Contd.

iii. *Consistency*:

- Requirements in SRS are said to be consistent if there are no conflicts between any set of requirements.
- Ex. of conflict include differences in terminologies used at separate places, logical conflicts like time period of report generation, etc.

iv. *Unambiguousness*:

- A SRS is said to be unambiguous if all the requirements stated have only 1 interpretation.
- Some of the ways to prevent unambiguousness include the use of modelling techniques like ER diagrams, proper reviews and buddy checks, etc.

Requirements Characteristics ..Contd.

v. *Ranking for importance and stability*:
   - There should a criterion to classify the requirements as less or more important or more specifically as desirable or essential.
   - An identifier mark can be used with every requirement to indicate its rank or stability.

vi. *Modifiability*:
   - SRS should be capable of easily accepting changes to the system to some extent.
   - Modifications should be properly indexed and cross-referenced.

Requirements Characteristics ..Contd.

vii. *Verifiability*:
- A SRS is verifiable if there exists a specific technique to quantifiably measure the extent to which every requirement is met by the system.
- Ex., a requirement stating that the system must be user-friendly is not verifiable and listing such requirements should be avoided.

viii. *Traceability*:
- One should be able to trace a requirement to design component and then to code segment in the program.
- One should be able to trace a requirement to the corresponding test cases.

ix. *Design Independence*:
- There should be an option to choose from multiple design alternatives for the final system.
- SRS should not include any implementation details.

Requirements Characteristics ..Contd.

x. *Testability*:
- A SRS should be written in such a way that it is easy to generate test cases and test plans from the document.

xi. *Understandable by the customer*:
- An end user maybe an expert in his/her specific domain but might not be an expert in computer science.
- Hence, the use of formal notations and symbols should be avoided to as much extent as possible.
- The language should be kept easy and clear.

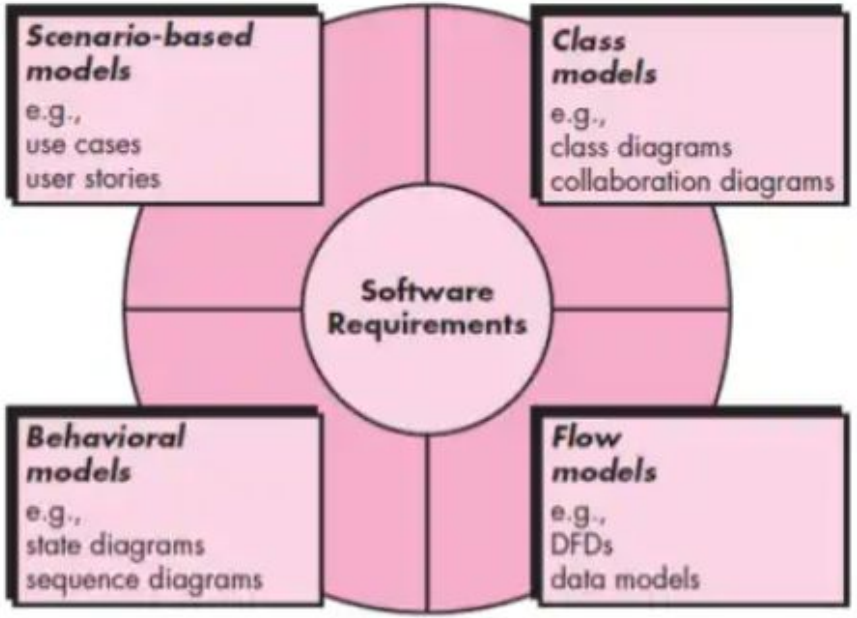
xii. *Right level of abstraction*:
- If the SRS is written for the requirements phase, the details should be explained explicitly.
- Whereas, for a feasibility study, fewer details can be used.
- Hence, the level of abstraction varies according to the purpose of the SRS.

Requirements Modeling

- A model is a representation of some aspect of the system being built.
- Requirements modeling is an approach used in projects where the requirements keep on evolving throughout the project.
  - This will help ensure that the project's exact requirements are documented clearly.
  - You can create a strong foundation for the project's overall needs and specifications.
  - Creating a model can help clarify and refine the design.
  - It helps to simplify complex aspects of systems.
- Models can be in a variety of graphical representations/ forms:
  - Structured approach
    - Entity Relationship Diagram (ERD)
    - Data Flow Diagram (DFD)
  - Object Oriented approach (UML Diagrams)
    - Use Case Diagram
    - Class Diagram
    - Sequence Diagram
    - Collaboration Diagram
    - Activity Diagram/ Swim lanes
    - State Diagram

Requirements Modeling

- The Requirements modeling action results in one or more of the following types of models:



Requirements Modeling

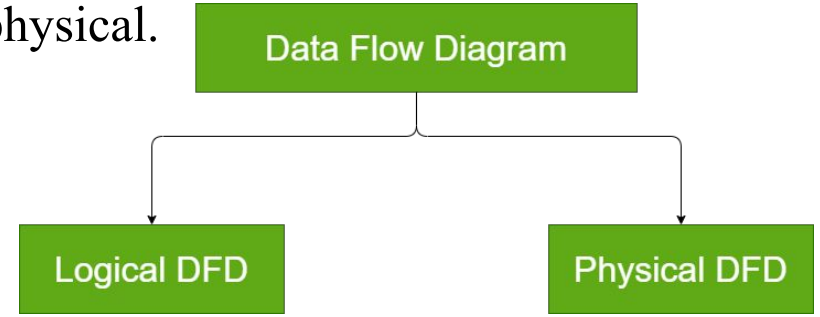
- The Requirements modeling action results in one or more of the following types of models:
  - Scenario-based models => they depict the system from the point of view of customer/ user (various system “actors”).
  - Data models => they depict the information domain for the problem.
  - Class-oriented models => they represent object-oriented classes (attributes and operations) and the manner in which classes collaborate to achieve system requirements.
  - Flow-oriented models => they represent the functional elements of the system and how they transform data as it moves through the system.
  - Behavioral models => they depict how the software behaves as a consequence of external “events”.

Data Flow Diagram (DFD)

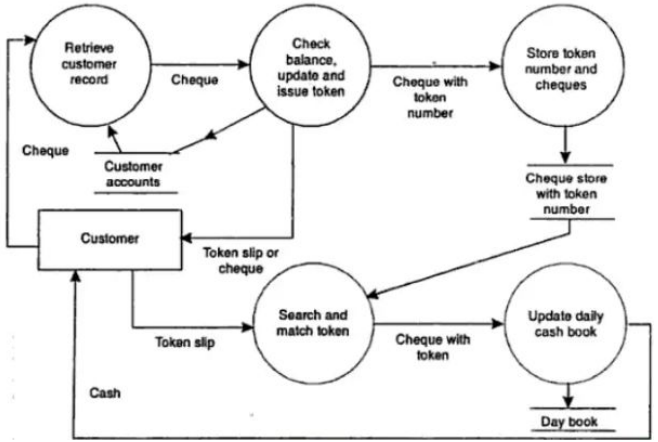
- A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system.
- A neat and clear DFD can depict the right amount of the system requirement graphically.
- It can be manual, automated, or a combination of both.
- It shows what kind of data will be input to the system and what data is received as the output.
- Also, it tells where the data will come from and go, what changes the information, and where it is stored in the process.
- 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.

Types of Data Flow Diagram

•DFDs are categorized as either logical or physical.

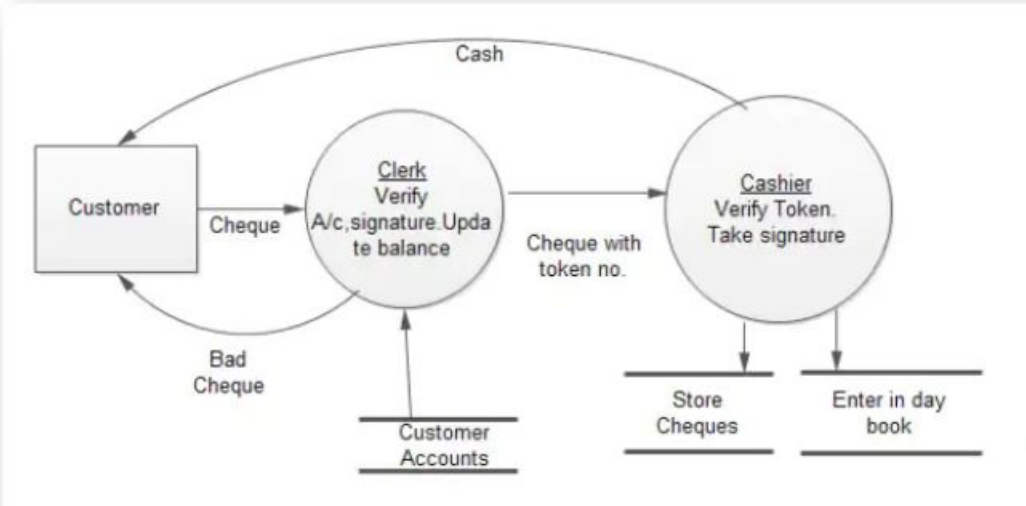


– Logical DFD:
  - It specifies the logical processes performed on the data.
  - It portrays system's activities.



Types of Data Flow Diagram …..

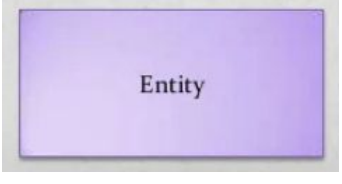
- Physical DFD:
  - It is a graphical representation of a system showing the system's internal and external entities, and the flows of data into and out of these entities.
  - It specifies who is actually carrying out the process.
  - It does not tell us what is being accomplished.



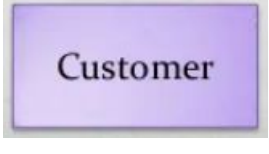
Elements of Data Flow Diagram

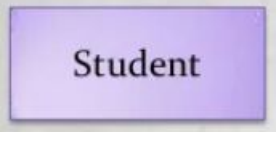
•Data Flow Diagrams are composed of four basic symbols:

- External Entity/ Entity/ Source or Sink
  - The sharp cornered rectangles (or simply boxes) in a DFD indicate entities.
  - External Entity symbol represents sources of data to the system or destinations of data from the system.
  - Source or Sink is an external entity and acts as a source of system inputs or sink of system outputs.



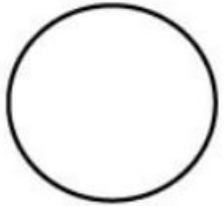
  - Entities are people, things, systems, sub systems, organizations, etc.

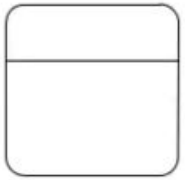




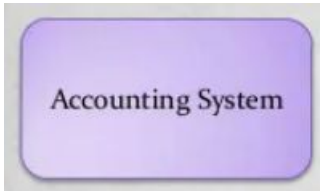
Elements of DFD …. Contd.

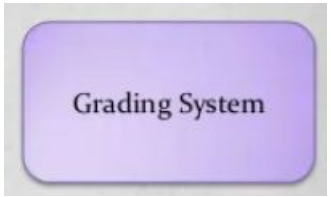
- Process
  - The rounded cornered rectangles or a circle/ bubble in a DFD indicate processes.





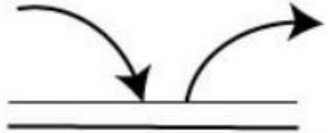
  - The process symbol represents an activity that transforms or manipulates the data (combines, reorders, converts, etc.)

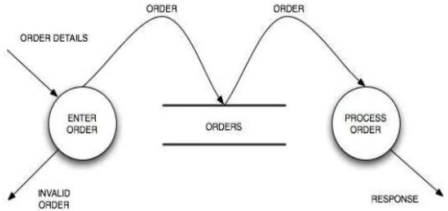




Elements of DFD …. Contd.

– Data Store
  - Open sided rectangles in DFD indicates data store.
  - A Data Store is a repository of data.

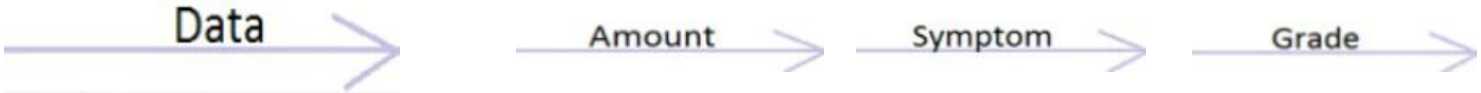




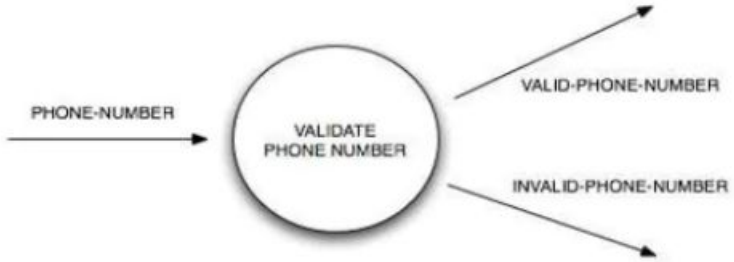
  - The Data Store symbol represents data that is not moving (delayed data at rest).
  - Data can be written into the data store. This is depicted by an incoming arrow.
  - Data can be read from a data store. This is depicted by an outgoing arrow.
  - External entity cannot read or write to the data store.
  - Two data stores cannot be connected by a data flow.

Elements of DFD …. Contd.

– Data Flow
  - A line with an arrow indicates data flow.



  - It shows the flow of data into or out of a process or a data store.
  - It represents movement of data.
  - A flow carries only one type of packet and is represented by the flow name.



Levels of DFD

- An entire system is depicted by a single data flow diagram which gives the complete system overview.
  - This is called **context diagram**.
- Successive expansion of processes is done to show the operations in details.
  - This is levelling of DFD.
- In the process of levelling the DFD, it still does not specify how the processing is being done, only the data flow is specified.

- DFD levels:
  1. Level 0 DFD
  2. Level 1 DFD
  3. Level 2 DFD

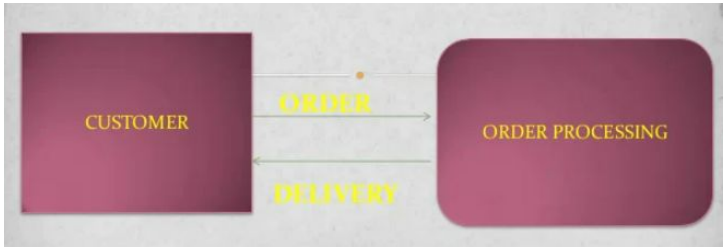
- The levels may continue if the system is very complex. However, generally the maximum level of DFD is 3.

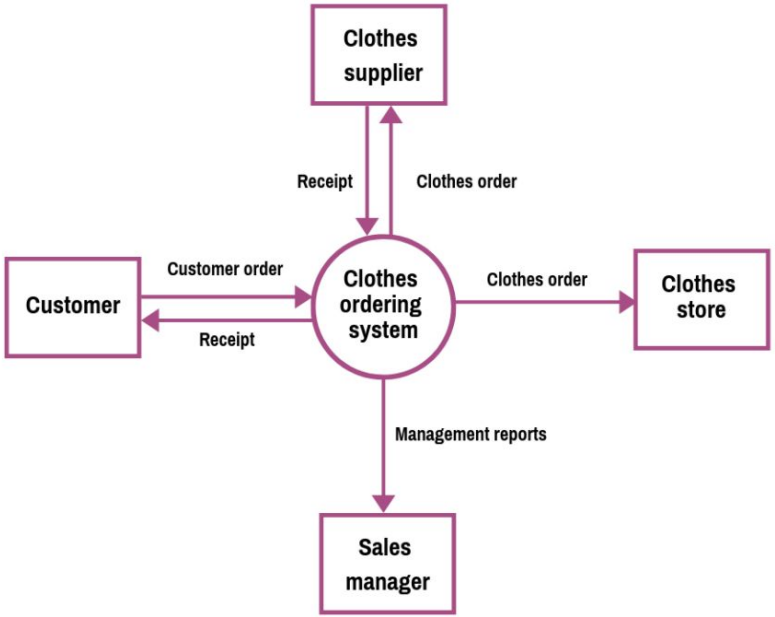
Level 0 DFD

- It is also known as context level diagram.
- It is the simplest DFD.
- The outermost level (0 level) is concerned with how the system interacts with the outside world.
- It shows the system as a single process with its relationship to external entities.
- It represents the entire system as a single bubble with input and output data indicated by incoming/outgoing arrows.

- How to create level 0 DFD:-
  - ✔ Identify the main system.
  - ✔ Identify the external people who interact with the system.
  - ✔ Decide what data these entities will enter into the system.
  - ✔ Determine what these entities expect as output from the system.

Level 0 DFD …. Contd.



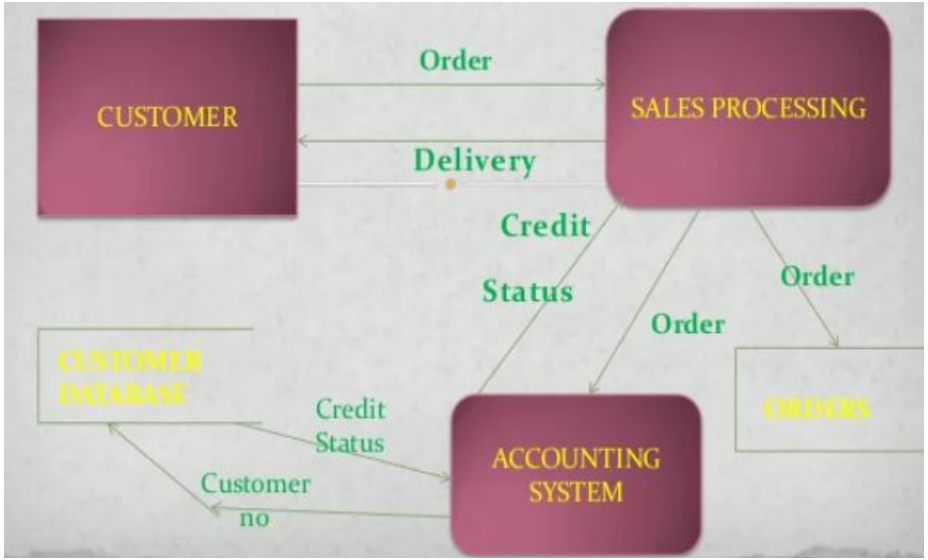


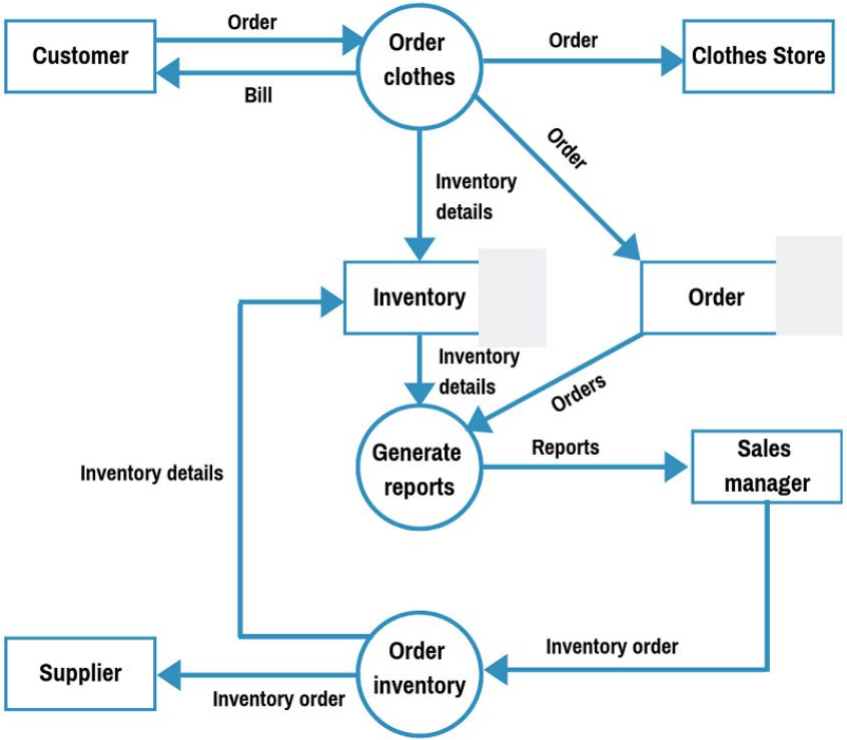
Level 1 DFD

- In 1-level DFD, the context diagram is decomposed into multiple bubbles/processes.
- The entire system is divided into modules. It represents how data moves through different modules.
- In this level, the main functions of the system are highlighted and breakdown the high-level process of 0-level DFD into subprocesses.
- It provides a high-level view of the system that identifies the major processes and data stores.

- How to create level 1 DFD:-
  - ✔ Focus on your process and break it into 2 or more sub-processes.
  - ✔ Identify what data flows between these processes and between the entities.
  - ✔ Identify what permanent data files are used in this system.
  - ✔ Note that no new entities can be introduced.

Level 1 DFD …. Contd.





Level 2 DFD

- It goes one step deeper into parts of 1-level DFD.
- Each process from level 1 is exploded even more into sub processes.
- This decomposition continues for each level.
- It can be used to plan or record the specific/necessary detail about the system's functioning.
- The number of levels possible depends on the complexity of the system.

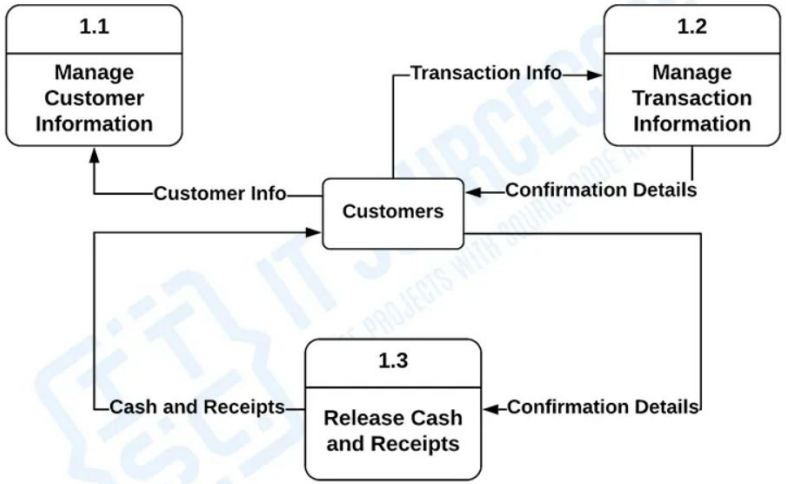
Value of DFD

- With a dataflow diagram, users are able to visualize how the system will operate, what the system will accomplish, and how the system will be implemented.

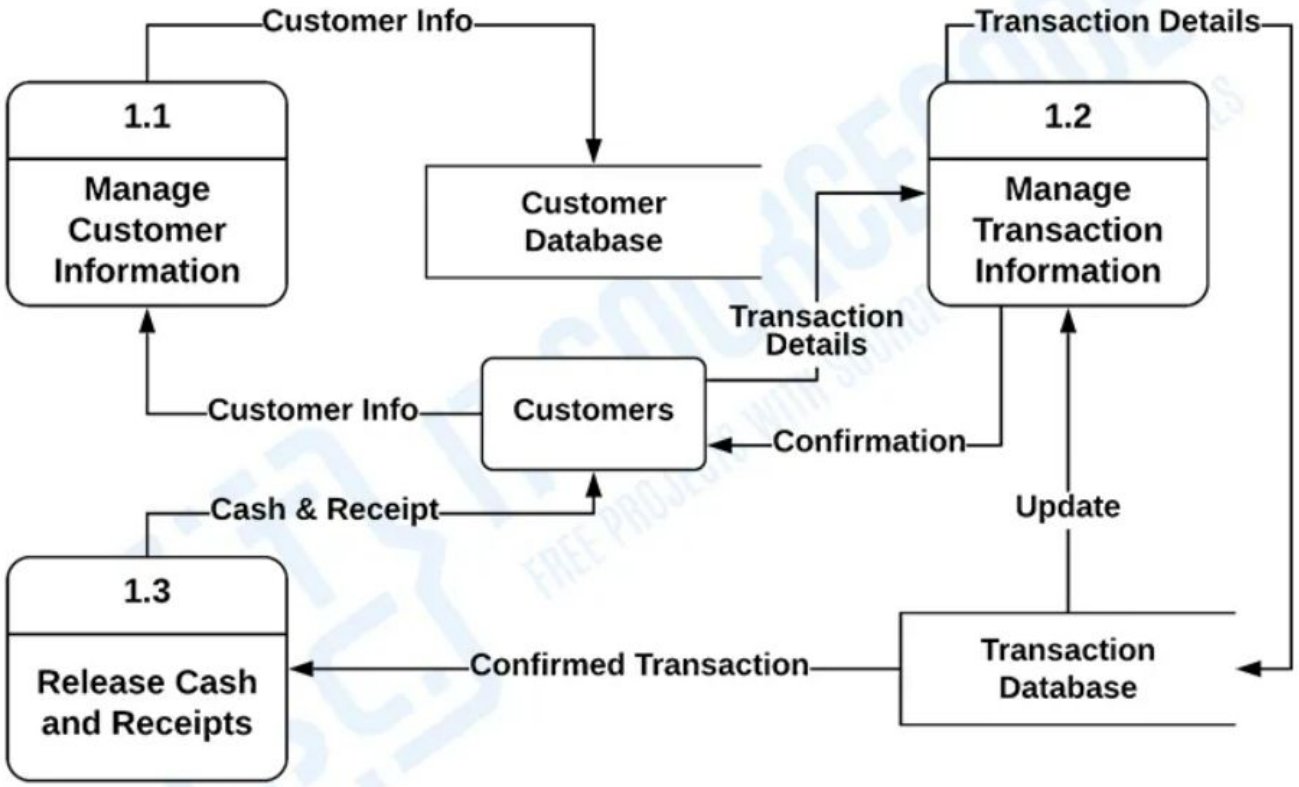
- Data flow diagrams can be used to provide the end users with the physical idea of how the data they input ultimately has an effect upon the structure of the whole system.

- The old system's dataflow diagrams can also be drawn up and compared with the new system's dataflow diagrams to draw comparisons in order to help implement a more efficient system.

Example of DFD (ATM System)



Example of DFD (ATM System)



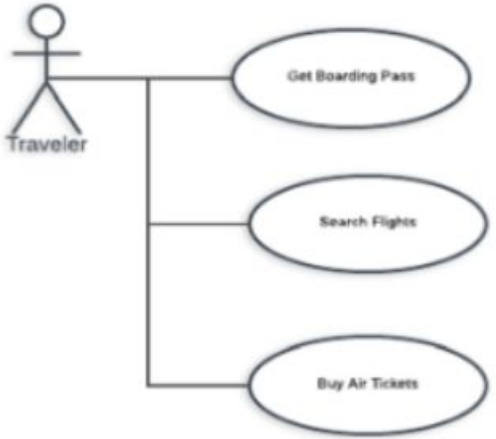
Scenario Based Model

- Requirements modeling comprises several stages, or '**patterns':**
  - scenario-based modeling,
  - data modeling,
  - flow-oriented modeling,
  - class-based modeling and
  - behavioral modeling.
- Each of these stages/patterns examines the same problem from a different perspective.
- Technically, there is no 'right way' of going through these stages, but generally, the process would begin with scenario-based modeling and complete with behavioral modeling.
- Scenario-based elements depict how the user interacts with the system and the specific sequence of activities that occur as the software is used.

Scenario Based Model ….Contd.

- Scenario-based modeling's primary objective is to look at a system from the user's perspective and produce a use case.
- Ex. An user interacting with the system, like the traveler using the airline application to generate their boarding pass.
- It makes sense to start with this step as the other requirements modeling stages/patterns will make reference to this use case.

- Following use case diagram depicts three cases of the traveler using the airline application.
  - The first is getting a boarding pass.
  - The next is searching for flights..
  - The last is buying air tickets.



Unified Modeling Language (UML)

- Use cases develop the understanding that how the system would be used.
- Hence, requirement modeling with UML (Unified Modeling Language) begins with the creation of scenarios in the form of use cases, activity diagrams and swim lanes.
- A single diagram cannot capture the different elements of a system in its entirety.
- Hence, UML is made up of nine diagrams that can be used to model a system at different point of time in the software life cycles of a system.
- Use case diagrams are used to gather the requirements of a system including internal and external influences.
- These requirements are mostly design requirements.
- Hence, when a system is analyzed to gather its functionalities, use cases are prepared and actors are identified.

Unified Modeling Language (UML)

- Use cases develop the understanding that how the system would be used.
- Hence, requirement modeling with UML (Unified Modeling Language) begins with the creation of scenarios in the form of use cases, activity diagrams and swim lanes.
- Use case diagrams are used to gather the requirements of a system including internal and external influences.
- These requirements are mostly design requirements.
- Hence, when a system is analyzed to gather its functionalities, use cases are prepared and actors are identified.
- Purpose of Use Case Diagram:
  - to gather the requirements of a system.
  - to get an outside view of a system.
  - identify the external and internal factors influencing the system.
  - show the interaction between the system and the actors.

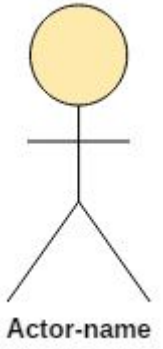
Use Case Diagram

- The Use Case diagram is used to identify the primary elements and process that form the system.

- Features/ functions of Use Case diagram:
  - Customer communicates with the developers and specifies his/her requirements.
  - Use Case diagram focuses on the behavior of the system from external (user's) point of view.
  - It helps developer to discover the requirements of the target system from the user's perspective.
  - It describes functions of the system from top to bottom.
  - Provides graphical representation/ description of the system.
  - Specifies types of interactions that take place within a system.
  - Defines boundary of the system.

Use Case Diagram …. Contd.

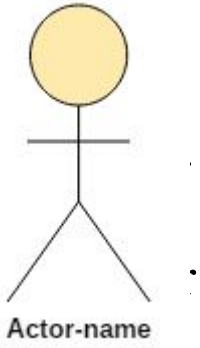
- Uses of Use Case diagram:
  - Design test cases for testing the functionality of the system.
  - Communication with the clients.
  - Generating new use cases.
  - Discovering new features/requirements.



- Elements of Use Case Diagram:
  a. Actor
     - It is an external entity that interacts with the system.
     - It interacts with the use cases.
     - An actor is someone outside the system boundary.

Elements of Use Case Diagram

- Actors are of two types:
  - Primary actor
    - » They are the main users/ entities of the system
    - » They are completely outside the system.
    - » They drive the system requirements.
    - » They use the system to achieve an observable
  - Secondary actor
    - » They are the users/entities that supervise, manage the system.
    - » They appear to be more inside the system than outside.

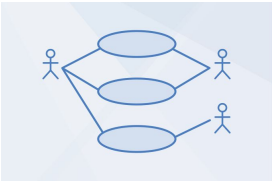


- Actors have unique names and roles/descriptions.
- Actor is denoted as a stickman.
- Ex. People, computer hardware and devices, external systems, etc.

Elements of Use Case Diagram …. Contd.

b. Use Case(s)

- A use case is a description of how a person who actually uses that process or system will accomplish a goal.
- It is typically associated with software systems, but can be used in reference to any process.
- They are the processes that describe the behavior of the system from the user's point of view.
- Use cases represent the sequence of events/ transactions that are performed by the actor or that take place within the system.

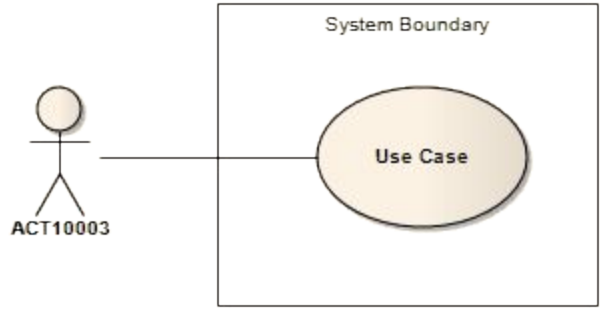


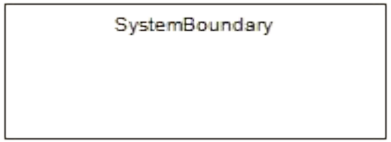
- An actor initiates a use case to assess the system functionality.
- The use case can then further initiate another use case and gather more information from the actor.
- It is represented as an oval/ ellipse with the name of the use case written inside the use case.
- Ex. Pay fees, book seat, perform operation, etc.

Elements of Use Case Diagram …. Contd.

c. System boundary

- It defines the limit of the system.
- It defines the scope of what a system will be.
- It encloses the use case.
- Use of system boundary helps to group logically related elements; from a visual perspective.
- It is denoted by a rectangular box.

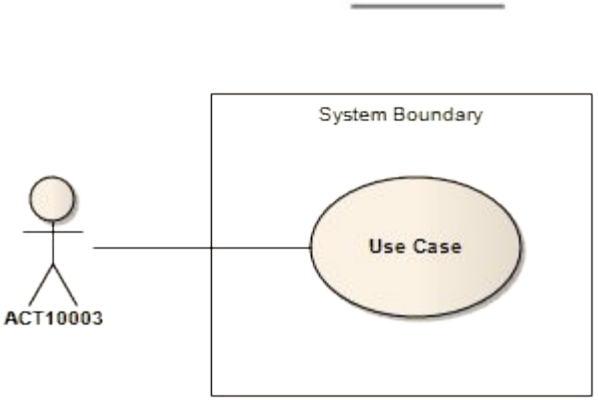


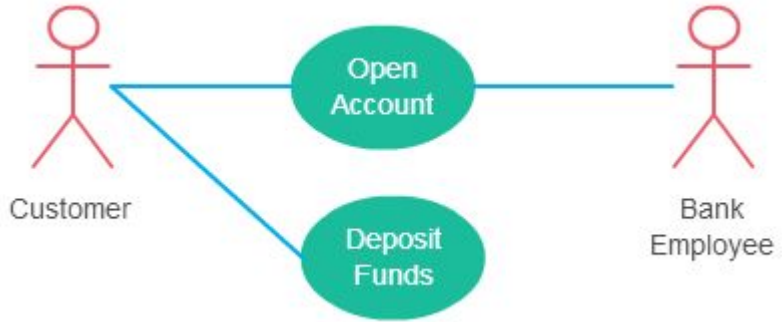


Elements of Use Case Diagram …. Contd.

d. Association

- Actor and use case can be associated to indicate that the actor participates in that use case.
- Thus, an association corresponds to a sequence of actions between the actor and use case in achieving the use case.
- It is represented as a line.





Elements of Use Case Diagram …. Contd.

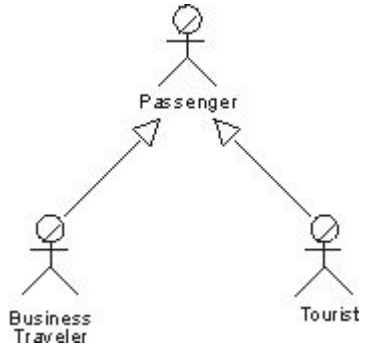
e. Relationship

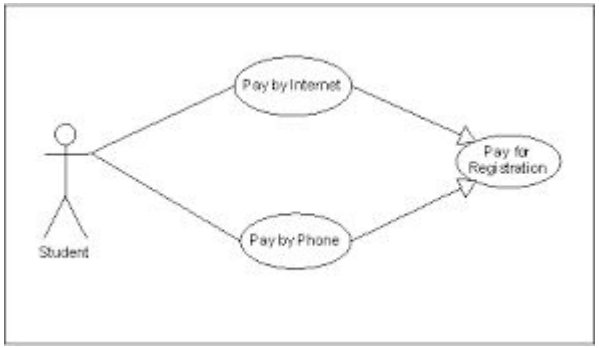
- It defines the relation between the use cases.
- It is of three types:

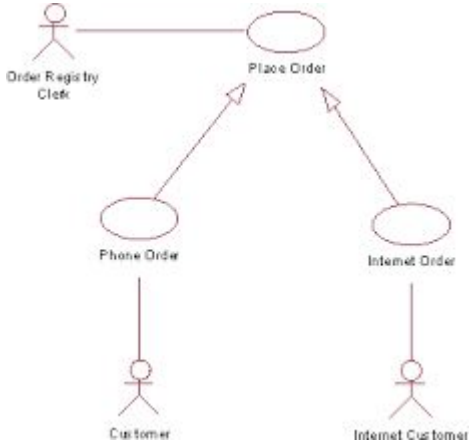
  - Generalization
    - » A generalization relationship is used to represent inheritance relationship between model elements of the same type.
    - » It has a parent-child relationship between use cases or actors.
    - » The child use case in the generalization relationship has the underlying business process meaning, but is an enhancement of the parent use case.
    - » In a use case diagram, generalization is shown as a directed arrow with a triangle arrowhead.
    - » The child use case is connected at the base of the arrow.
    - » The tip of the arrow is connected to the parent use case.

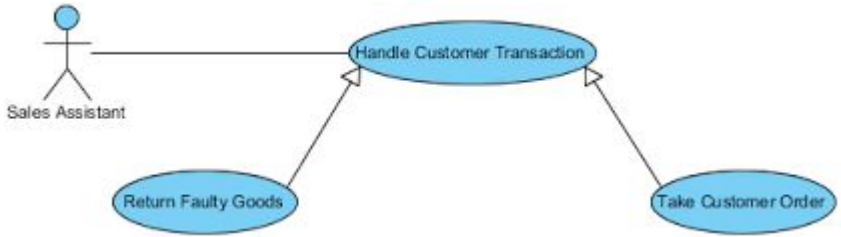
Elements of Use Case Diagram …. Contd.

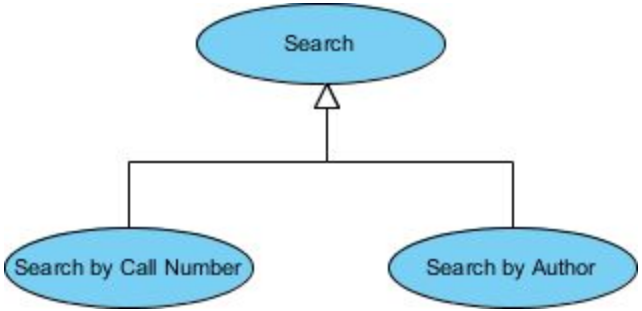
Generalization Relationship









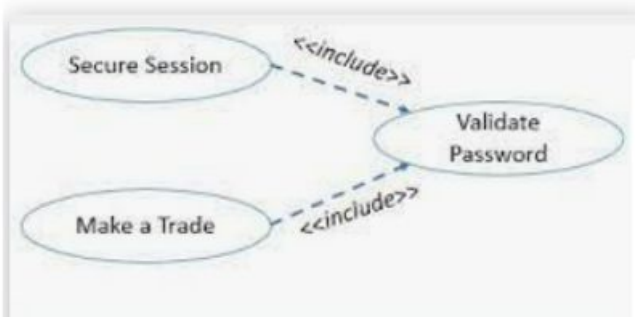


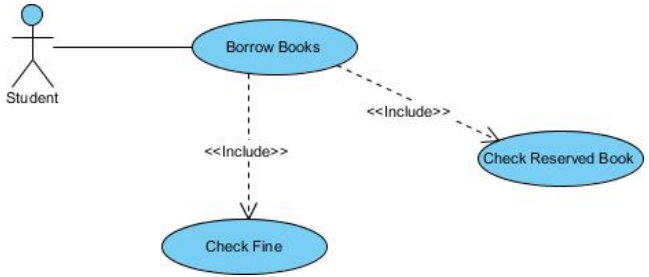
Elements of Use Case Diagram …. Contd.

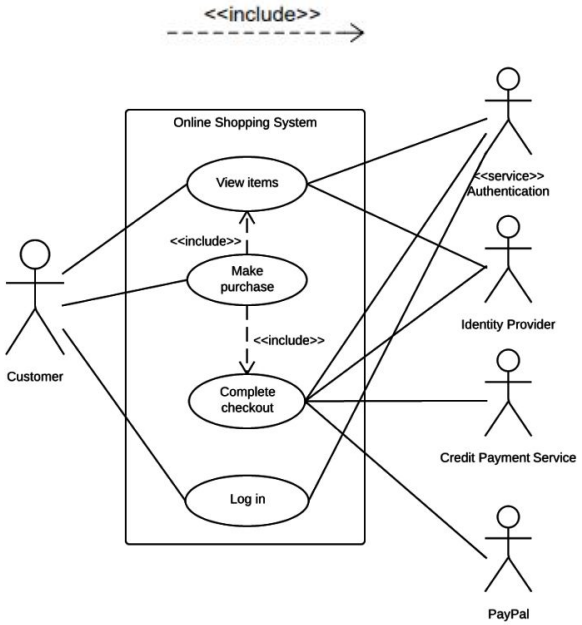
e. Relationship
   - Include
     - » Include relationship show that the behavior of the included use case is part of the including (base) use case.
     - » Include relationship helps to reuse common actions across multiple use cases.
     - » This is done to simplify complex behaviors.
     - » Few things to consider when using the <<include>> relationship.
       - The base use case is incomplete without the included use case.
       - The included use case is mandatory and not optional.

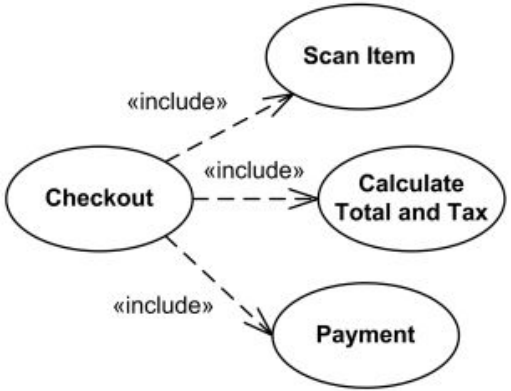
Elements of Use Case Diagram …. Contd.

- Include Relationship
  - » It is denoted as a dotted line an arrow head pointing from the parent use case to the child use case.



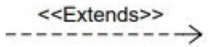






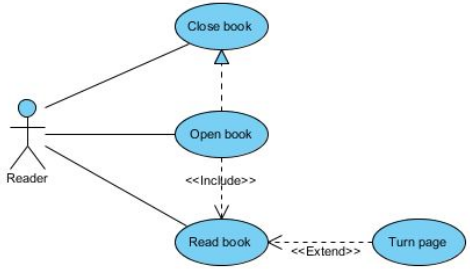
Elements of Use Case Diagram …. Contd.

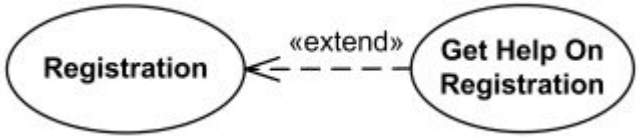
e. Relationship
   - Extend
     - » In an extend relationship between two use cases, the child use case adds to the existing functionality and characteristics of the parent use case.
     - » An extend relationship is depicted with a directed arrow having a dotted shaft, similar to the include relationship.
     - » The tip of the arrowhead points to the parent use case and the child use case is connected at the base of the arrow.
     - » The stereotype "<<extend>>" identifies the relationship as an extend relationship.

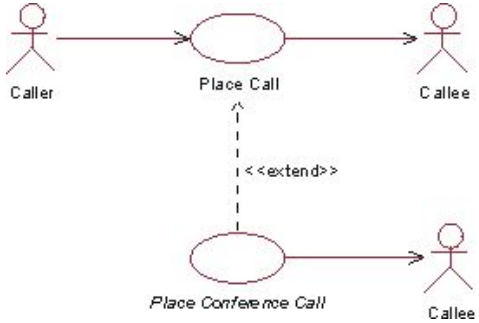


Elements of Use Case Diagram …. Contd.

- Extend
  - » Here are a few things to consider when using the <<extend>> relationship.
    - o It is dependent on the base/parent use case.
    - o The extending use case is usually optional and can be triggered conditionally.
    - o The parent use case must be meaningful on its own. i.e. it should be independent and must not rely on the behavior of the extending use case.





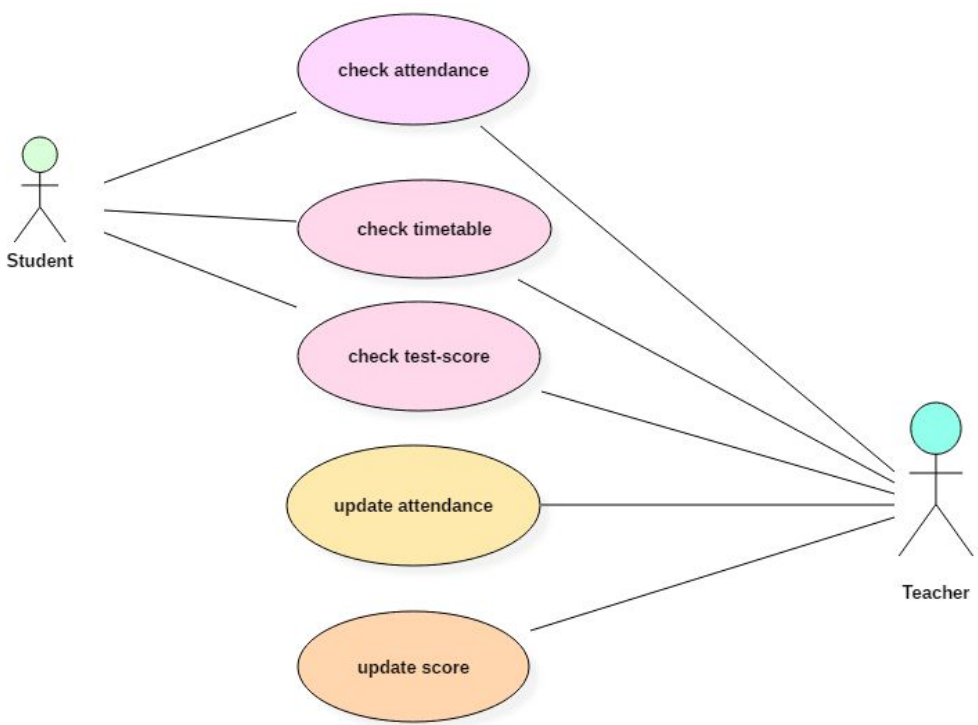


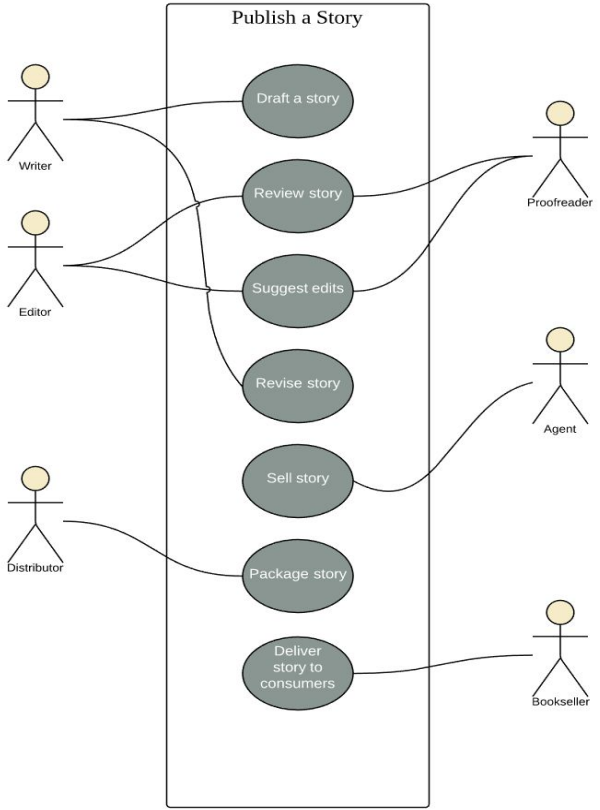
How to draw a Use Case Diagram

- Use case diagrams are considered for high level requirement analysis of a system.
- When the requirements of a system are analyzed, the functionalities are captured in use cases.
- Following items should be identified :
  - Functionalities to be represented as use case
  - Actors
  - Relationships among the use cases and actors.
- Use the following guidelines to draw an efficient use case diagram =>
  - The name of use case should be chosen in such a way so that it can identify the functionalities performed.
  - Give a suitable name for actors.
  - Show relationships and dependencies clearly in the diagram.
  - Do not try to include all types of relationships, as the main purpose of the diagram is to identify the requirements.

  - https://www.youtube.com/watch?v=zid-MVo7M-E

Example of Use Case Diagram





Class Diagram

- The class diagram depicts a static view of an application.
  - It represents the types of objects residing in the system and the relationships between them.
  - A class consists of its objects, and also it may inherit from other classes.
  - A class diagram is used to visualize, describe, document various different aspects of the system, and also construct executable software code.
  - It shows the attributes, classes, functions, and relationships to give an overview of the software system.
  - It constitutes class names, attributes, and functions in a separate compartment that helps in software development.
- Since it is a collection of classes, interfaces, associations, collaborations, and constraints, it is termed as a structural diagram.

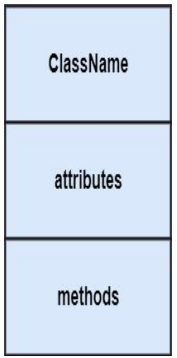
Class Diagram …. Contd.

- Purpose of Class Diagrams:
  - It is the only diagram that is widely used for construction, and it can be mapped with object-oriented languages.
  - It analyses and designs a static view of an application.
  - It describes the major responsibilities of a system.
  - It is a base for component and deployment diagrams.
  - It incorporates forward and reverse engineering.
- Benefits of Class Diagrams:
  - It can represent the object model for complex systems.
  - It reduces the maintenance time by providing an overview of how an application is structured before coding.
  - It provides a general schematic of an application for better understanding.
  - It represents a detailed chart by highlighting the desired code, which is to be programmed.
  - It is helpful for the stakeholders and the developers.

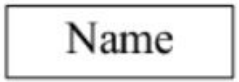
Elements of Class Diagram

a. Class:

- In UML, a class represents an object or a set of objects that share a common structure and behavior.
- It is represented either as a rectangular box or a rectangular box with three rows/ sections.



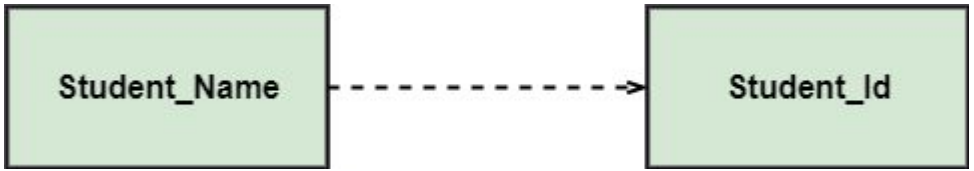
- The upper section encompasses the name of the class.
  - The initial letter of class name should always begin with a capital letter.
- The middle section constitutes the attributes, which describe the quality of the class.
- The lower section contain methods or operations. It demonstrates how a class interacts with data.



Elements of Class Diagram …. Contd.

b. Relationships:

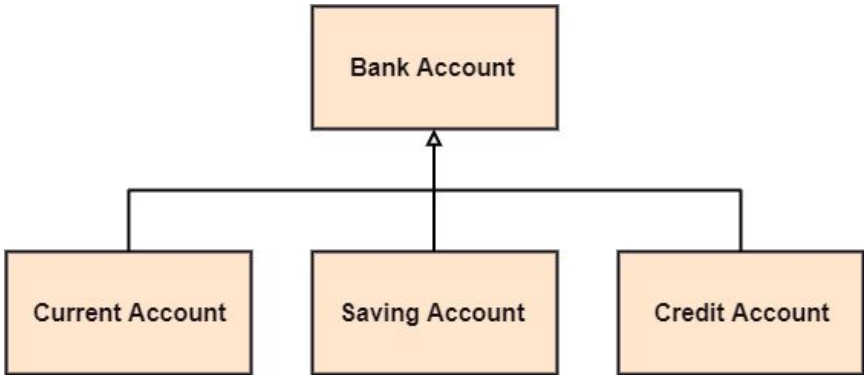
- In UML, relationships are of three types:
  - Dependency:
    - A dependency is a semantic relationship between two or more classes where a change in one class cause changes in another class.
    - It forms a weaker relationship.
    - Ex. Student_Name is dependent on the Student_Id.



Elements of Class Diagram …. Contd.

b. Relationships:

- 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.
  - Ex. Current Account, Saving Account, and Credit Account are the generalized form of Bank Account.



Elements of Class Diagram …. Contd.

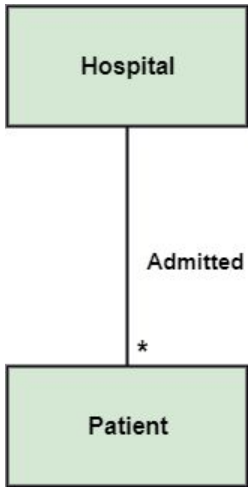
b. Relationships:

- Association:
  - It describes a static or physical connection between two or more objects.
  - It depicts how many objects are there in the relationship.
  - Ex. a department is associated with the college.

Elements of Class Diagram …. Contd.

c. Multiplicity:

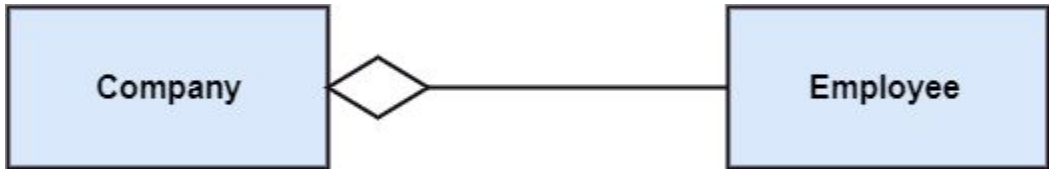
- It defines a specific range of allowable instances of attributes.
- In case if a range is not specified, one is considered as a default multiplicity.
- Ex. multiple patients are admitted to one hospital.



Elements of Class Diagram …. Contd.

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



Elements of Class Diagram …. Contd.

d. 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.
- Ex. A contact book consists of multiple contacts, and if you delete the contact book, all the contacts will be lost.

- Ex. Sales order system

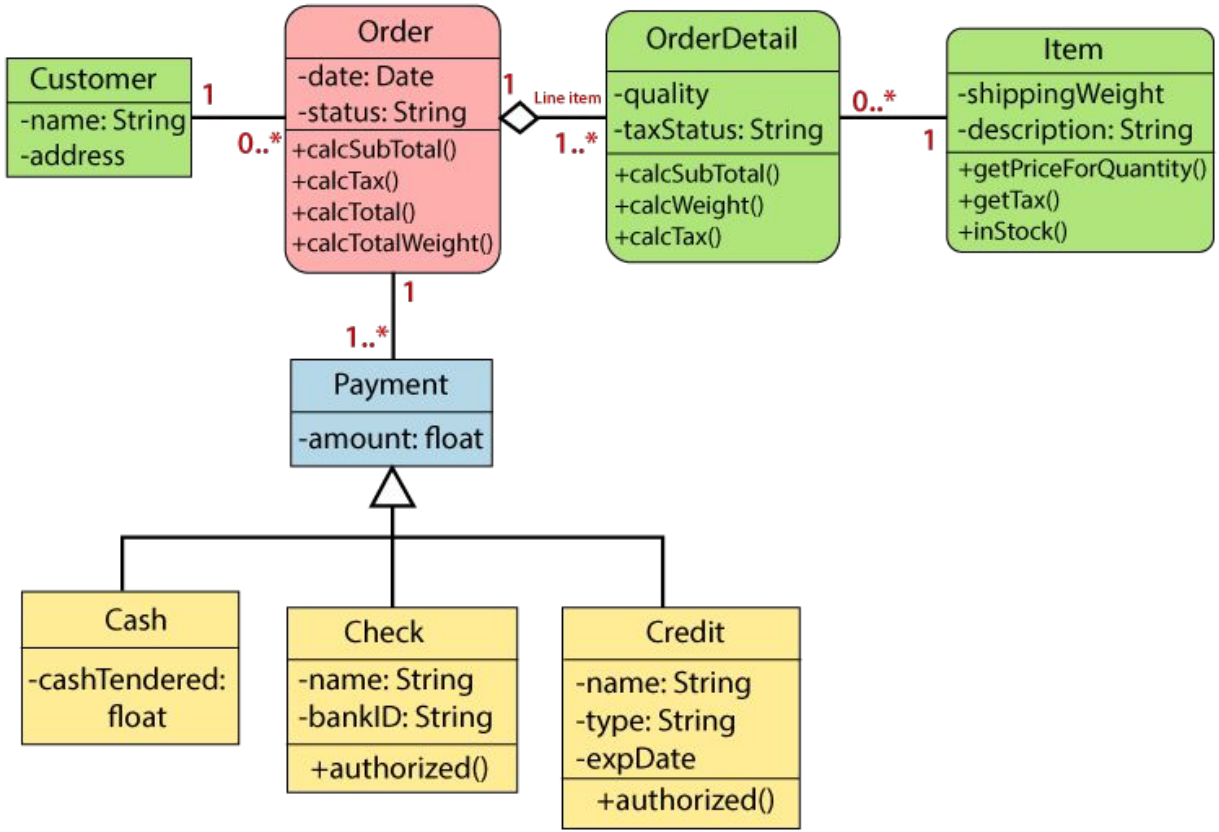
How to draw a Class Diagram

- Class diagram is basically a graphical representation of the static view of the system and represents different aspects of the application.
- A collection of class diagrams represent the whole system.
- The following points should be remembered while drawing a class diagram –
  - The name of the class diagram should be meaningful to describe the aspect of the system.
  - Each element and their relationships should be identified in advance.
  - Responsibility (attributes and methods) of each class should be clearly identified
  - For each class, minimum number of properties should be specified, as unnecessary properties will make the diagram complicated.

  - https://www.youtube.com/watch?v=UI6lqHOVHic

Example of Class Diagram

- Ex. Sales order system



Sequence Diagram

- A sequence diagram is the most commonly used interaction diagram.
  - An interaction diagram is used to show the interactive behavior of a system.
  - Since visualizing the interactions in a system can be a cumbersome task, different types of interaction diagrams are used to capture various features and aspects of interaction in a system.
- A sequence diagram is a Unified Modeling Language (UML) diagram.
- It simply depicts interaction between objects in a sequential order i.e. the order in which these interactions take place.
- Sequence diagrams describe how and in what order the objects in a system function.
- These diagrams are widely used by businessmen and software developers to document and understand requirements for new and existing systems.

Sequence Diagram …. Contd.

- 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.

- Benefits of a Sequence Diagram:
  - It explores the real-time application.
  - It depicts the message flow between the different objects.
  - It has easy maintenance.
  - It is easy to generate.
  - It implements both forward and reverse engineering.
  - It can easily update as per the new change in the system.

Elements of Sequence Diagram

a. Class/ Object:

- Class is represented in the similar manner as a class in the class diagram.
- They represent the various objects that are participants in the sequence diagram.
- Object is denoted as a rectangular box consisting of the object name which is underlined.
- The object name always begins with lower case along with a colon.

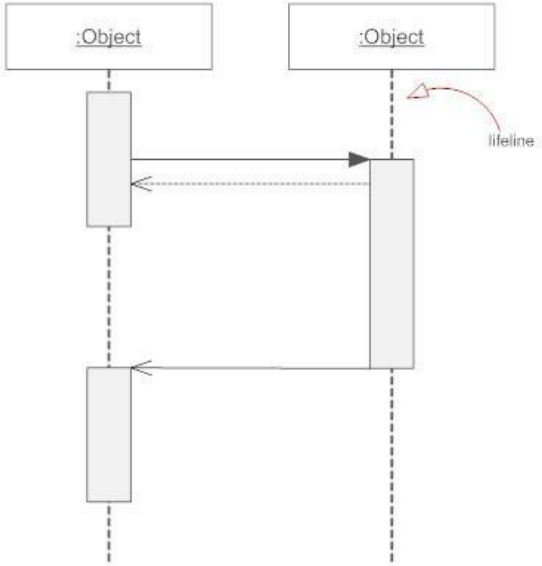
Elements of Sequence Diagram …. Contd.

b. Actor:

- 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.
- An actor exists outside the system and is not a part of the system.
- Actors depict various roles including human users and other external subjects.
- It is represented using a stick person notation.
- We can have multiple actors in a sequence diagram.

Elements of Sequence Diagram …. Contd.

c. Lifeline(s):
   - Lifelines are vertical dashed lines that indicate the object's presence over time.
   - They represent the life of an object in the scenario.



Elements of Sequence Diagram …. Contd.

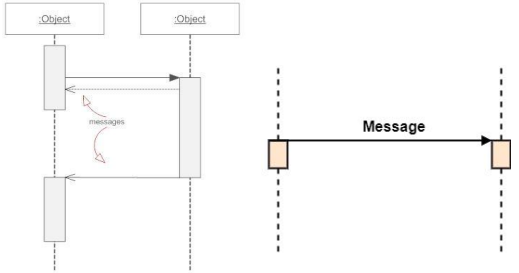
d. Activation:

- Activation boxes represent the time an object needs to complete a task.
- 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.
- Hence, when an object is busy executing a process or waiting for a reply message, use a thin gray rectangle placed vertically on its lifeline.

Elements of Sequence Diagram …. Contd.

e. Messages:

- Communication between objects is depicted using messages.
- The messages appear in a sequential order on the lifeline.
- Messages are represented using arrows. Lifelines and messages form the core of a sequence diagram.



- Types of messages
  - Synchronous message
    - A synchronous message requires a response before the interaction can continue.
    - It is usually drawn using a line with a solid arrowhead pointing from one object to another.

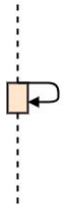
Elements of Sequence Diagram …. Contd.

- Asynchronous message
  - Asynchronous messages do not need a reply for interaction to continue.
  - Like synchronous messages, they are drawn with an arrow connecting two lifelines; however, the arrowhead is usually open and there is no return message depicted.

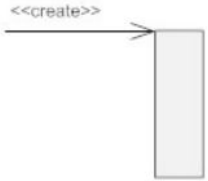
- Reply or Return message
  - A reply message is drawn with a dotted line and an open arrowhead pointing back to the original lifeline.

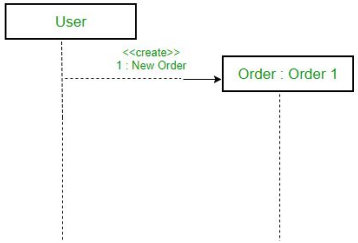
Elements of Sequence Diagram …. Contd.

- Self message
  - A message an object sends to itself, usually shown as a U shaped arrow pointing back to itself.



- Create message
  - This is a message that creates a new object.
  - Similar to a return message, it is depicted with a dashed line and an open arrowhead that points to the rectangle representing the object created.



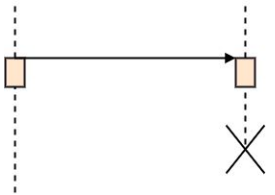


Elements of Sequence Diagram …. Contd.

- Delete message
  - This is a message that destroys an object.
  - It can be shown by an arrow with an x at the end.

g. Destroy object:

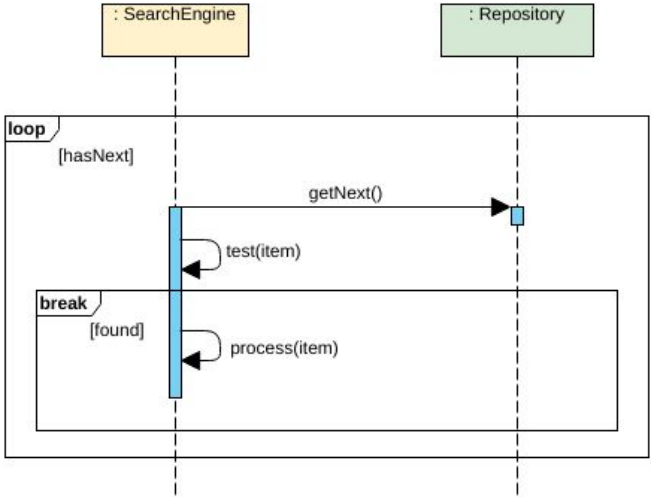
- Objects can be terminated early using an arrow labeled "<< destroy >>" that points to an X.
- This object is removed from memory.
- When that object's lifeline ends, you can place an X at the end of its lifeline to denote a destruction occurrence.



Elements of Sequence Diagram …. Contd.

g. Loops:

- A repetition or loop within a sequence diagram is depicted as a rectangle.
- Place the condition for exiting the loop at the bottom left corner in square brackets [ ].



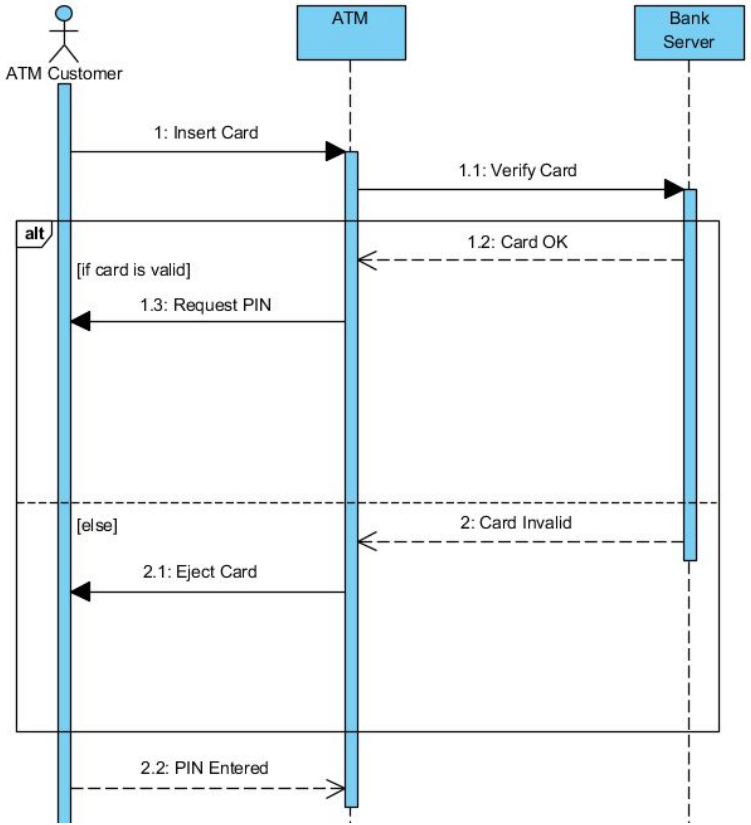
How to draw a Sequence Diagram

- The sequence diagram captures the time sequence of the message flow from one object to another.
- Following things are to be identified clearly before drawing the interaction diagram:
  - Objects taking part in the interaction.
  - Message flows among the objects.
  - The sequence in which the messages are flowing.
  - Object organization.

- https://www.youtube.com/watch?v=pCK6prSq8aw

Example of Sequence Diagram

- ATM System



Collaboration Diagram

- Collaboration diagram is an interaction diagram.
- It is also known as communication diagram.
  - It captures dynamic behavior of the objects in the system (message oriented).
  - Unlike a sequence diagram, a collaboration diagram shows the relationships among the objects. Every message is numbered.
  - It is very useful for visualizing the relationship between objects collaborating to perform a particular task.
  - You can have objects and actor instances in collaboration diagrams, together with links and messages describing how they are related and how they interact.
  - The diagram describes what takes place in the participating objects, in terms of how the objects communicate by sending messages to one another.

Collaboration Diagram …. Contd.

- Purpose of Collaboration Diagram:
  - To model flow of control by time sequence.
  - To depict relationship between the objects.
  - To illustrate coordination of object structure and control.
  - To describe the structural organization of the objects.
  - To describe the interaction among objects.

- Benefits of a Collaboration Diagram:
  - It is useful for analyzing use cases.
  - It depicts the message flow between the different objects.
  - It implements both forward and reverse engineering.
  - It is very useful for visualizing the relationship between objects collaborating to perform a particular task.
  - It can help you determine the accuracy of your static model (i.e., class diagrams).

Elements of Collaboration Diagram

a. Object:
   - The interaction between objects takes place in a system.
   - An object is depicted by a rectangle with the name of the object, preceded by a colon and underline.

b. Actor:
   - Actors are instances that invoke the interaction in the diagram.
   - Each actor has a name and a role, with one actor initiating the entire use case.

Elements of Collaboration Diagram ….

c. Link/ Association/ Relation:

- 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.

Elements of Collaboration Diagram ….

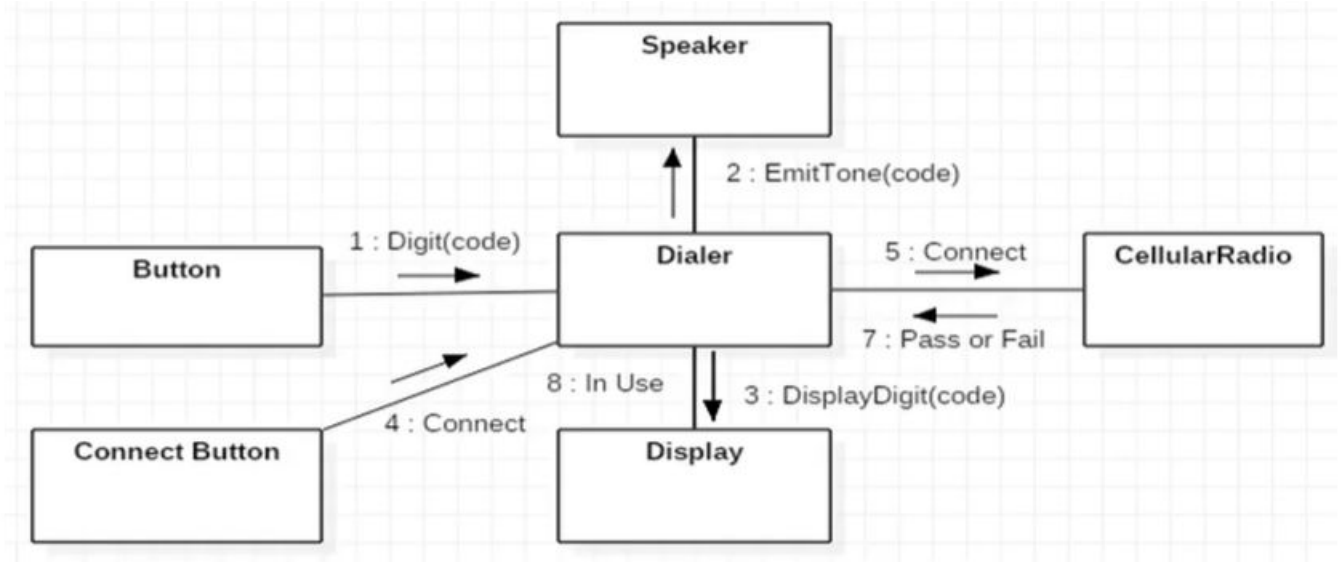
d. Message:

- It is a communication between objects which carries information.
- The sequence or order of the interaction is depicted by the 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.

How to draw a Collaboration Diagram

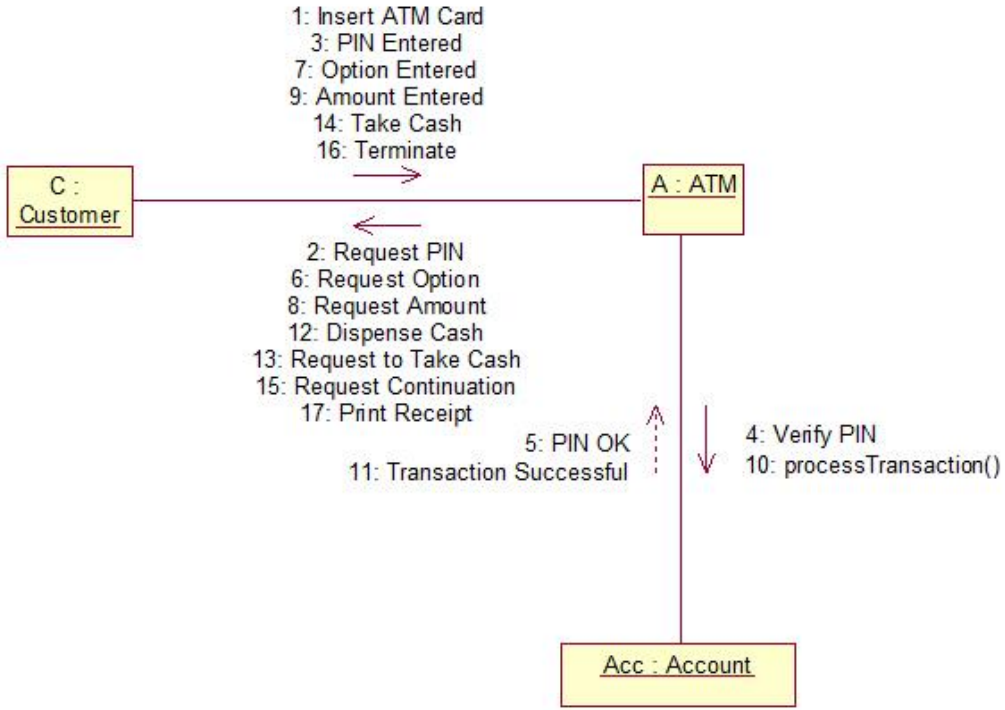
- The collaborations are used when it is essential to depict the relationship between the object.
- The collaboration diagrams are best suited for analyzing use cases.
- Steps for creating a Collaboration Diagram:
  - Determine the behavior for which the realization and implementation are specified.
  - Discover the structural elements that are class roles, objects, and subsystems for performing the functionality of collaboration.
  - Choose the context of an interaction: system, subsystem, use case, and operation.
  - Think through alternative situations that may be involved.
  - A specification level diagram may be made in the instance level sequence diagram for summarizing alternative situations.

Example of Collaboration Diagram



Example of Collaboration Diagram

Ex. ATM System



Activity Diagram

- Activity is a particular operation of the system. An activity diagram is a behavioral diagram i.e. it depicts the behavior of a system.
- It portrays the control flow from a start point to a finish point showing the various decision paths that exist while the activity is being executed.
- It demonstrates the flow of control within the system rather than the implementation.
- It helps in envisioning the workflow from one activity to another. It puts emphasis on the condition of flow and the order in which it occurs.
- The flow can be sequential, branched, or concurrent, and to deal with such kinds of flows, the activity diagram has come up with a fork, join, etc.
- It is also termed as an object-oriented flowchart. It encompasses activities composed of a set of actions or operations that are applied to model the behavioral diagram.

Activity Diagram …. Contd.

- Activity diagrams are not only used for visualizing the dynamic nature of a system, but they are also used to construct the executable system by using forward and reverse engineering techniques.

- Purpose of Activity Diagram:
  - Draw the activity flow of a system.
  - Describe the sequence from one activity to another.
  - Describe the parallel, branched and concurrent flow of the system.

- Benefits of a Activity Diagram:
  - Dynamic modelling of the system or a process.
  - Illustrate the various steps involved in a UML use case.
  - Model software elements like methods, operations and functions.
  - Show the constraints, conditions and logic behind algorithms.
  - To depict concurrent activities easily.

Elements of Activity Diagram

a. Activity/ Activities:

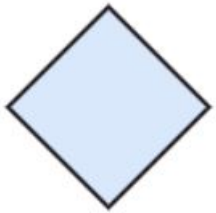
- The categorization of behavior into one or more actions is termed as an activity.
- It can be said that an activity is a network of nodes that are connected by edges.
- The edges depict the flow of execution. It may contain action nodes, control nodes, or object nodes.
- The control flow of activity is represented by control nodes and object nodes that illustrates the objects used within an activity.
- The activities are initiated at the initial node and are terminated at the final node.

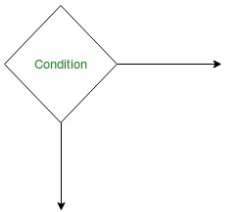
Elements of Activity Diagram …. Contd.

b. Initial Activity:
   - It depicts the initial stage or beginning of the set of actions/ activities.

c. Final Activity:
   - It is the stage where all the control flows and object flows end.

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





Elements of Activity Diagram …. Contd.

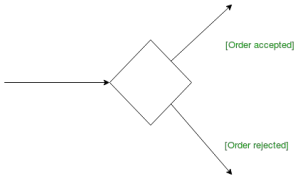
e. Transition/ Control Flow/ Action Flow:

- Action flows or Control flows are also referred to as paths and edges.
- They are used to show the transition from one activity state to another.
- An activity state can have multiple incoming and outgoing action flows.
- We use a line with an arrow head to depict a Control Flow.
- If there is a constraint to be adhered to while making the transition it is mentioned on the arrow.

Elements of Activity Diagram …. Contd.

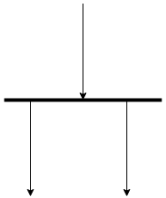
f. Guards/ Guard Condition:

- A Guard refers to a statement written next to a decision node on an arrow sometimes within square brackets.



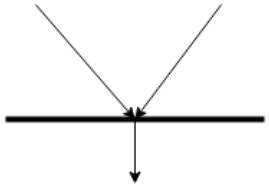
g. Fork:

- Fork nodes are used to support concurrent activities.
- It is used when both the activities get executed concurrently i.e. no decision is made before splitting the activity into two parts. Both parts need to be executed in case of a fork statement.
- We use a rounded solid rectangular bar to represent a Fork notation with incoming arrow from the parent activity state and outgoing arrows towards the newly created activities.

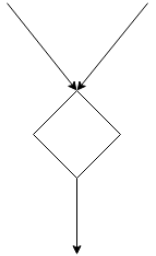


Elements of Activity Diagram …. Contd.

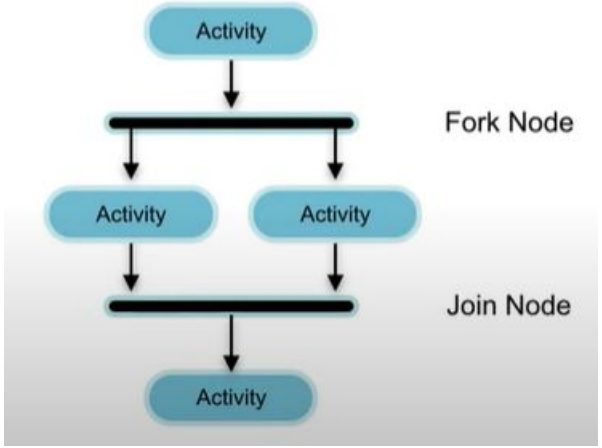
h. Join:
- Join nodes are used to support concurrent activities converging into one.
- For join notations we have two or more incoming edges and one outgoing edge.

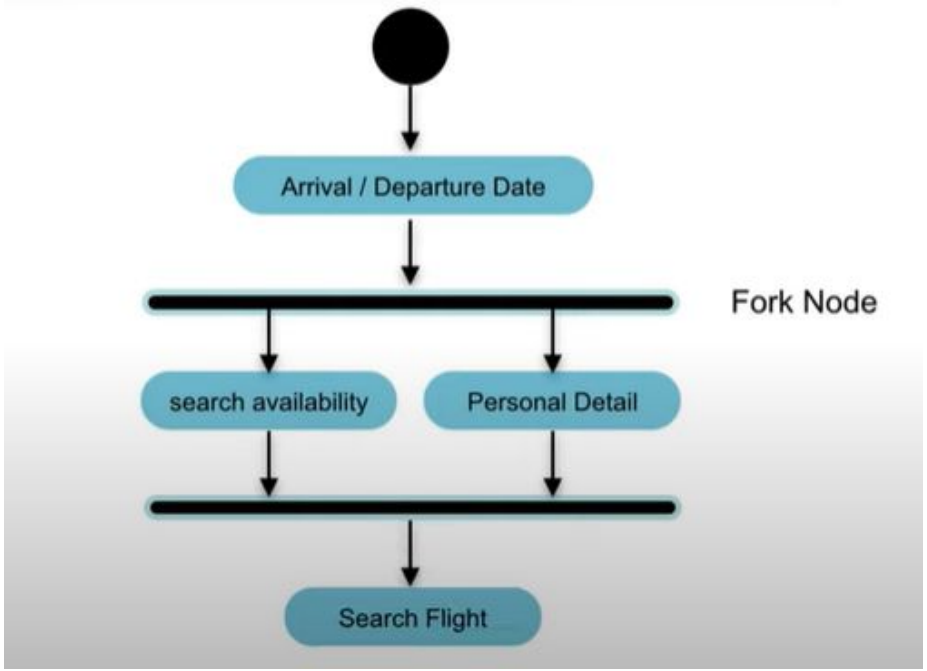


i. Merge/ Merge Event:
- Scenarios arise when activities which are not being executed concurrently have to be merged.
- We can merge two or more activities into one if the control proceeds onto the next activity irrespective of the path chosen.
- Merge notation is used for such scenarios.

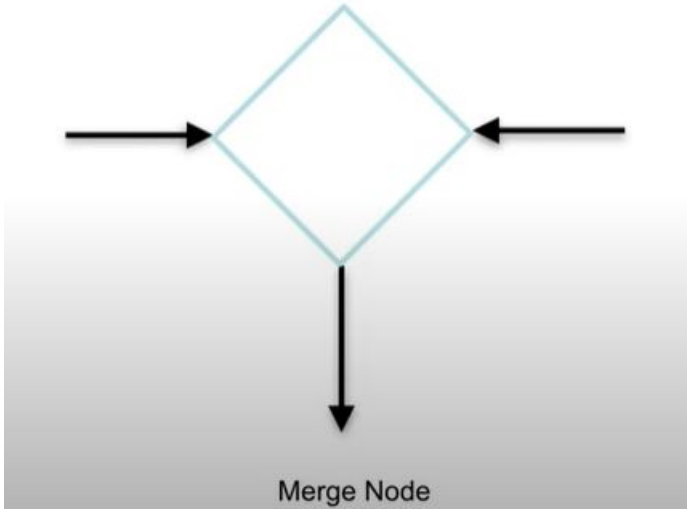


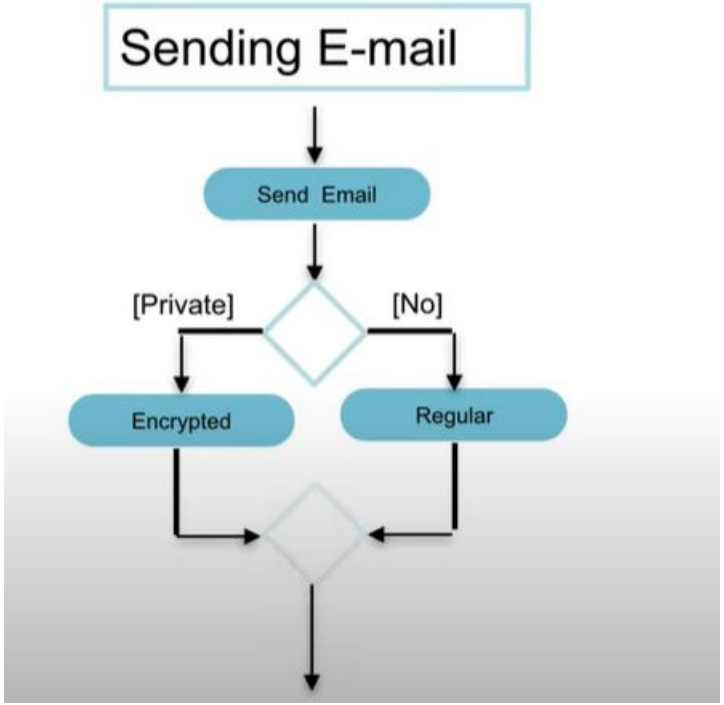
Elements of Activity Diagram …. Contd.





Elements of Activity Diagram …. Contd.





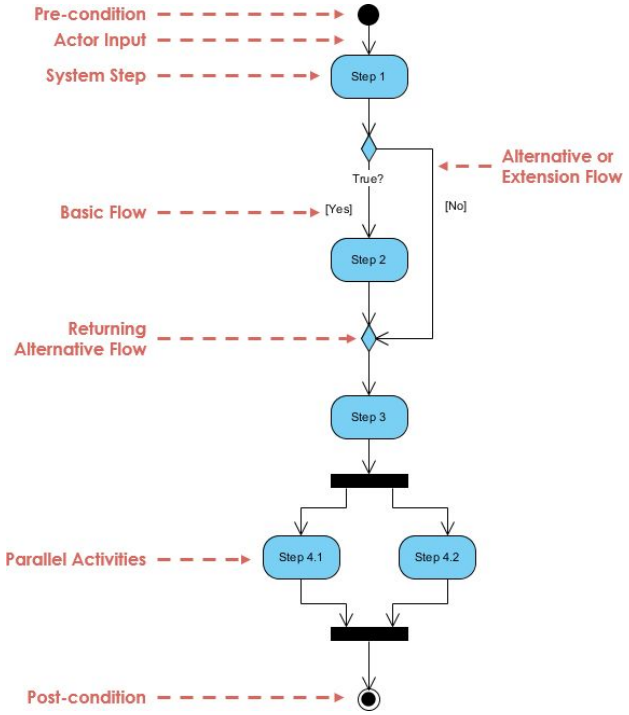
How to draw a Activity Diagram

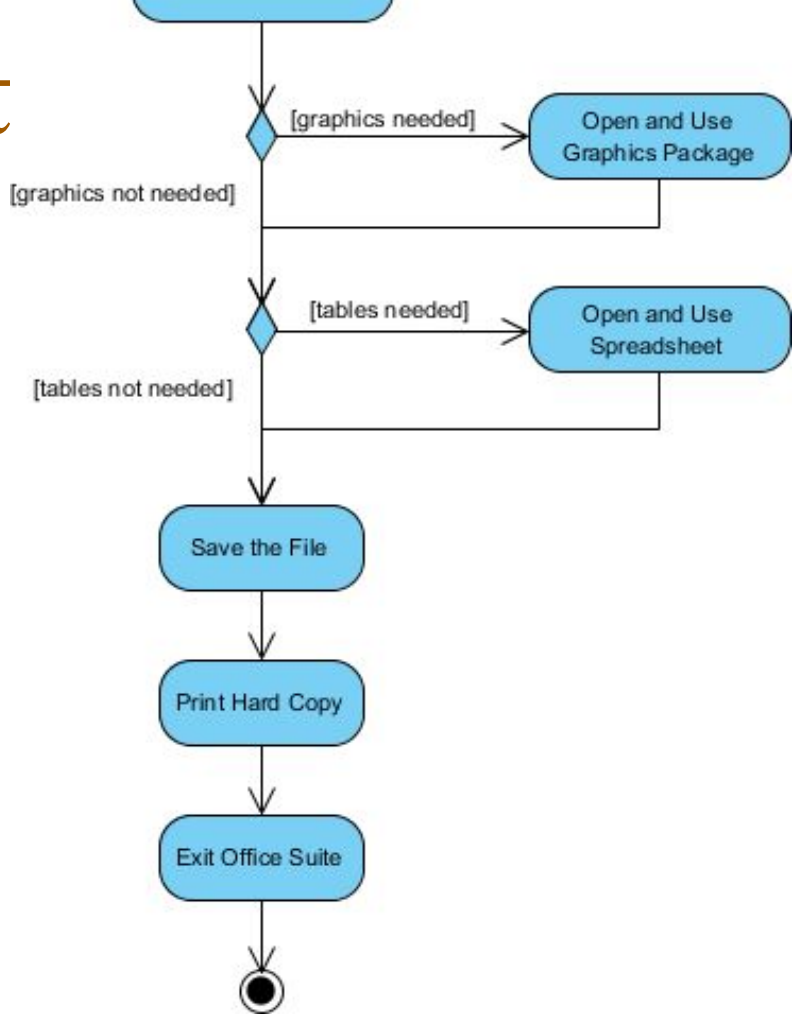
- An activity diagram is a flowchart of activities, as it represents the workflow among various activities.
- They are identical to the flowcharts, but they themself are not exactly the flowchart.
- An activity diagram is an enhancement of the flowchart, which encompasses several unique skills.

- Following are the rules that are to be followed for drawing an activity diagram:
  - A meaningful name should be given to each and every activity.
  - Identify all of the constraints.
  - Acknowledge the activity associations.

  - https://www.youtube.com/watch?v=3Hw_VXea73o

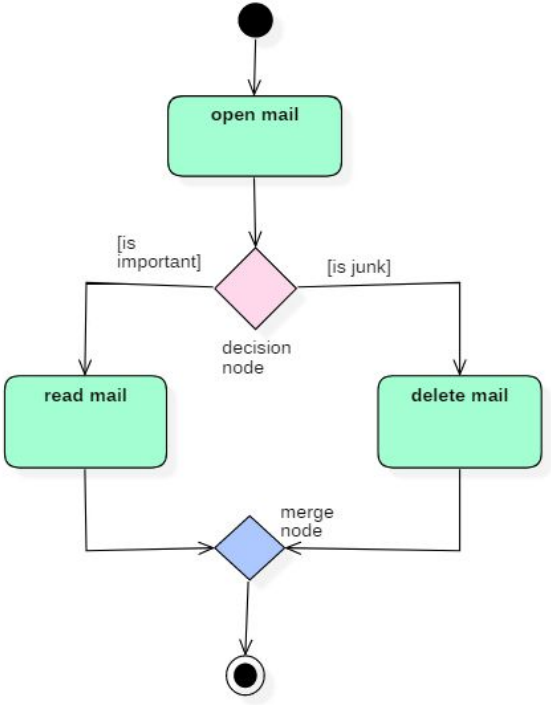
Steps for Act



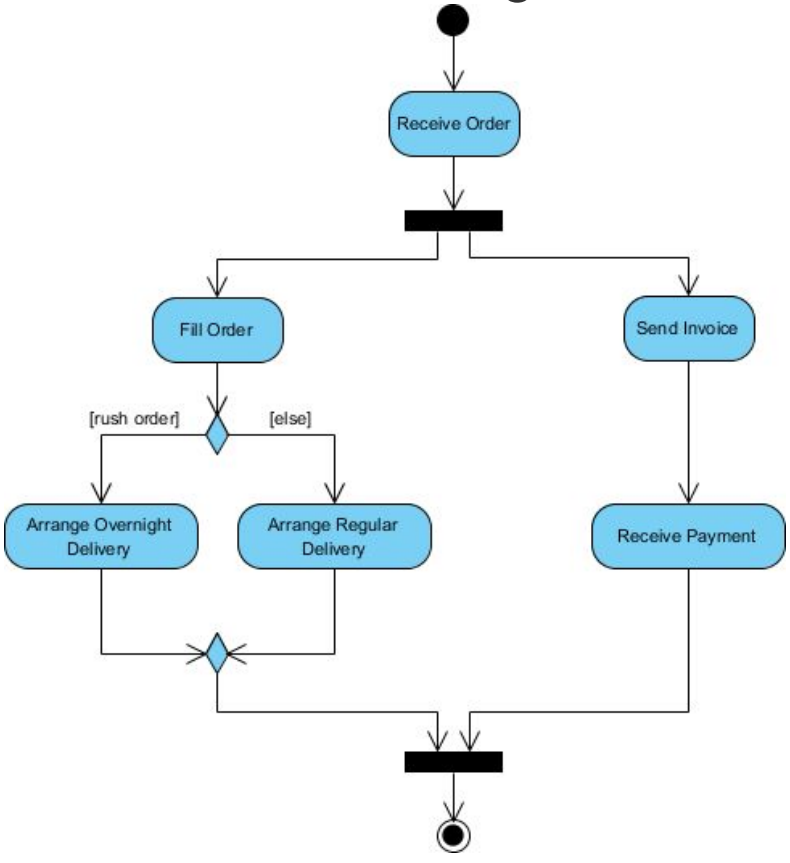


Example of Activity Diagram

Ex. Processing an email

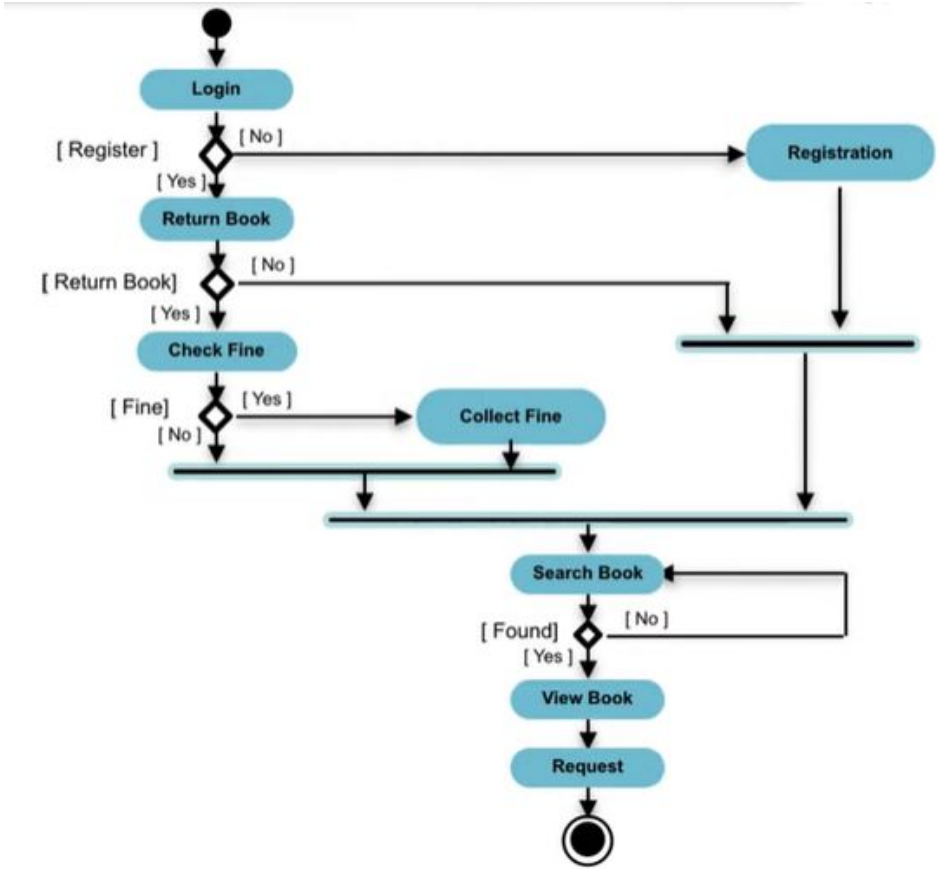


Order Processing



Example of Activity Diagram

Ex. Borrow Book



Swimlane Diagram

- Swimlanes are a part and parcel of Activity diagram.
- They show who is responsible for each part of a process.
- The name references the lanes we see in a swimming pool, with each lane acting as a space for a business role or department.
- They combine the activity diagram's depiction of logic with the interaction diagram's depiction of responsibility.
- Swimlanes are also called Functional Bands, Multi-Column Charts, or Rummler-Brache diagrams.
- They may sound complicated, but in practice swimlane diagrams are an elegant type of flowchart used to un-complicate diagrams that would otherwise be overly complex.
- Swimlanes have to be ordered in a Logical Manner.
- It is suggested to have less than five swimlanes in an activity diagram.

Need for Swimlane Diagram

- There are various benefits of using swim lane diagrams:
  i. Clarify complex processes
     - Inefficiencies and redundancies usually happen due to repeated or wasted efforts.
     - This happens due to their subsequent inability to identify their responsibilities.
     - This is easily resolved, as the visual nature of Swimlane diagrams makes it hard to ignore and necessary to identify these issues, discuss them, and organize them, as they become a part of the overall process improvement.

  ii. Improved communication and understanding
     - By creating Swimlane diagrams for different activities, it becomes easier for stakeholders and the employees to better understand the process.
     - This will save the resources and keep employees happy.

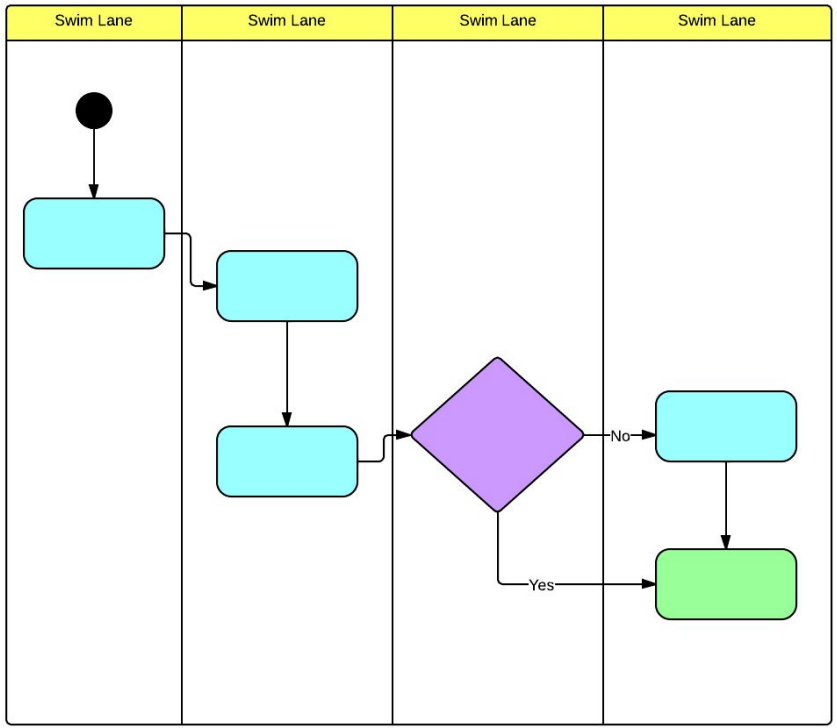
Need for Swimlane Diagram ….Contd.

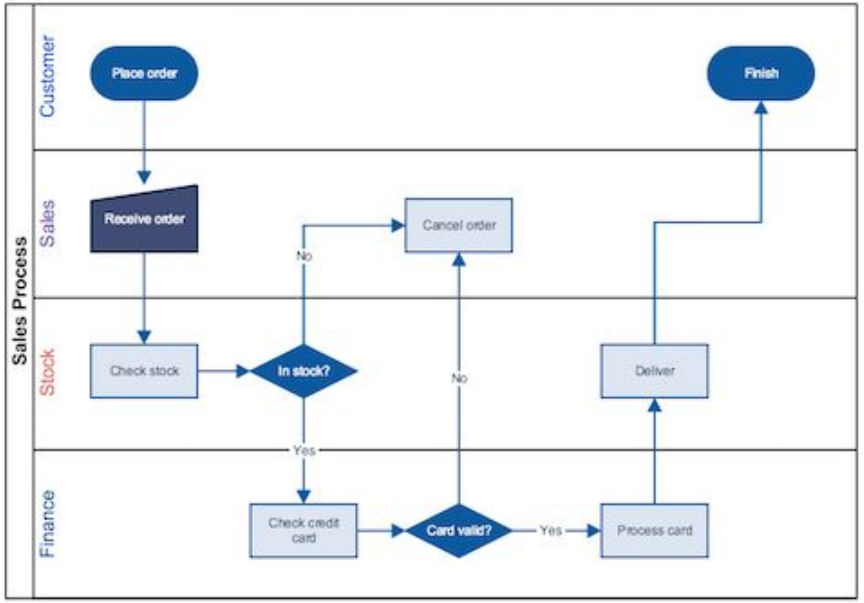
iii. Point out the participants
- Through Swimlane diagram, one can identify the participants and their sequence in the process.
- The input and output of these entities are also identified.

iv. Flexibility
- Swimlane diagrams can rotate, as they can be drawn both vertically and horizontally.
- You can proceed to horizontally place information in row sequences, and assign the roles vertically in columns, or the other way around.
- It can help to create different perspectives and emphasize details and roles.

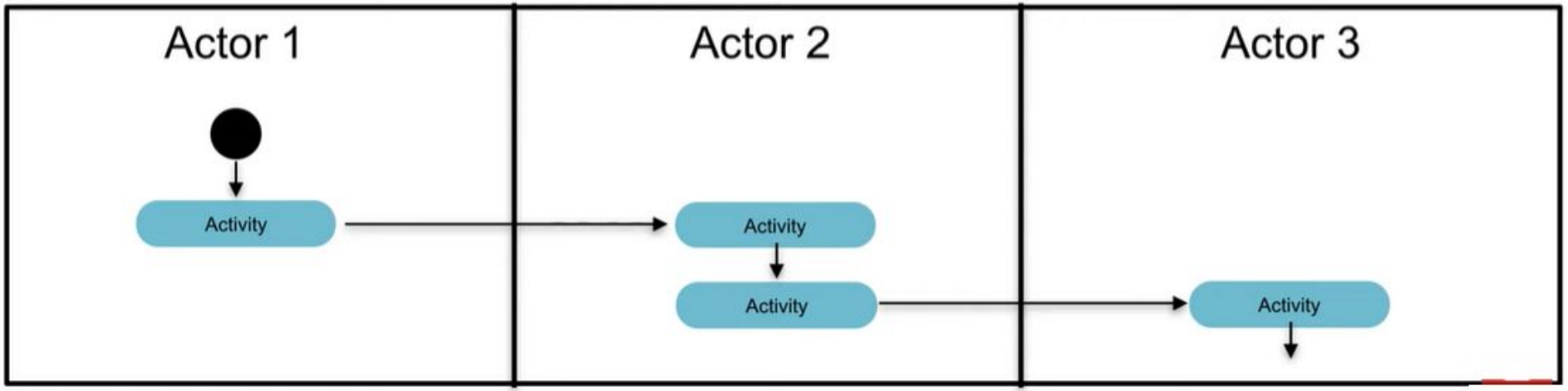
v. Easy analysis, continuous improvement
- One can check for any errors that might occur through the process without waiting for its completion.
- As Swimlane diagrams provide a detailed view of a business process they can help you in your efforts for continuous process improvement.

Sample of Swimlane Diagram

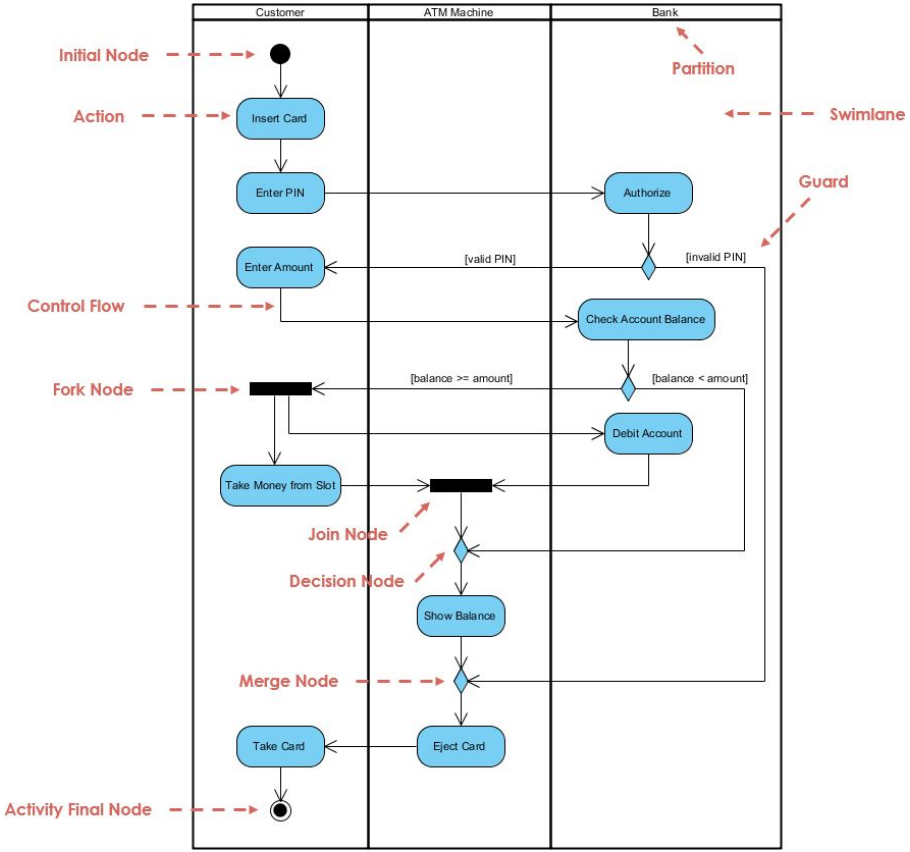


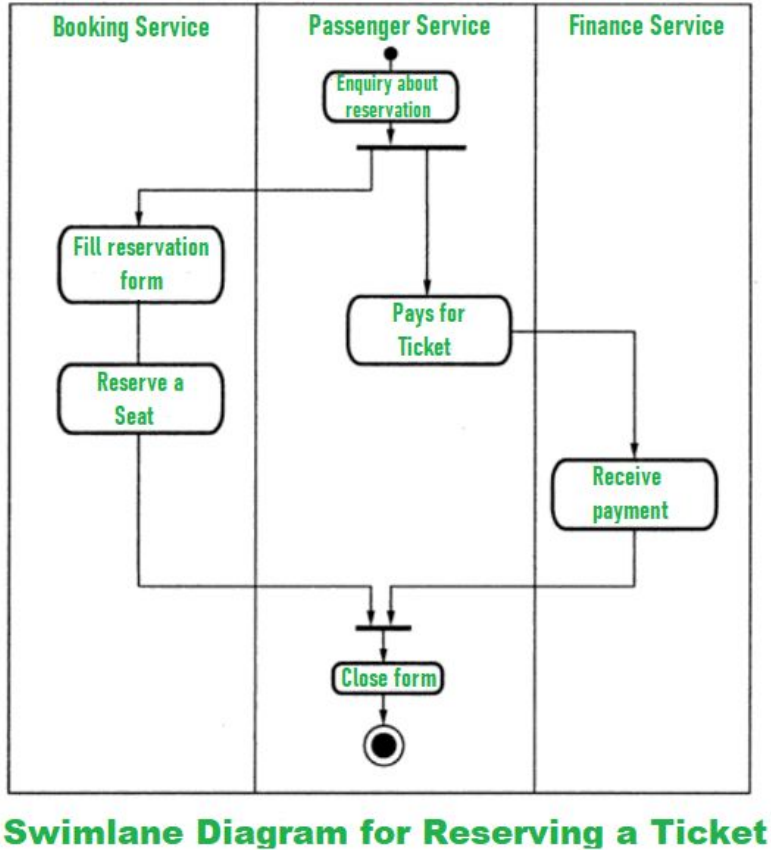


Sample of Swimlane Diagram



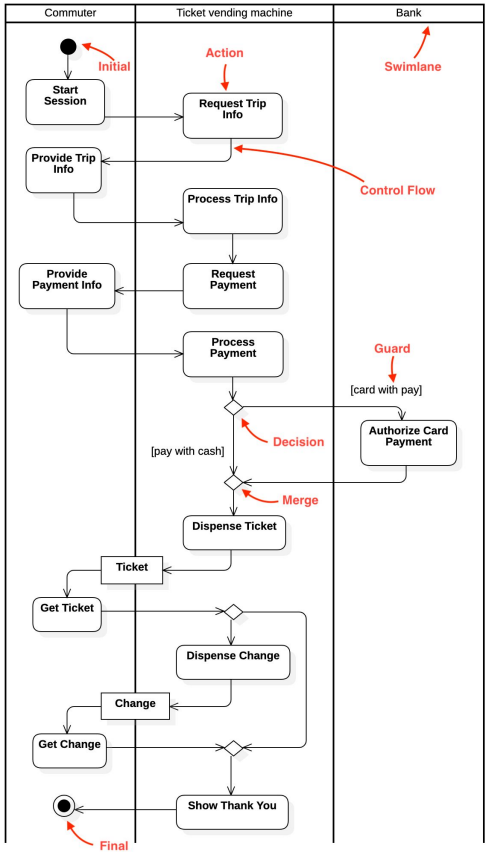
Example of Swimlane Diagram





Swimlane Diagram for Reserving a Ticket

Example of Swimlane Diagram



State Diagram

- The state machine diagram is also called the Statechart or State Transition diagram.
- It shows the order of states underwent by an object within the system.
- It captures the software system's behavior.
- It models the behavior of a class, a subsystem, a package, and a complete system.
- It tends out to be an efficient way of modeling the interactions and collaborations in the external entities and the system.
- It models event-based systems to handle the state of an object.
- It also defines several distinct states of a component within the system. Each object/component has a specific state.
- It blueprints an interactive system that response back to either the internal events or the external ones.
- The execution flow from one state to another is represented by a state machine diagram.
- It visualizes an object state from its creation to its termination.

State Diagram …. Contd.

- Statechart diagram defines different states of an object during its lifetime and these states are changed by events.
- It describes the flow of control from one state to another state. States are defined as a condition in which an object exists and it changes when some event is triggered.

- Purpose of State Diagram:
  - To model the dynamic aspect of a system.
  - To model the life time of a reactive system.
  - To describe different states of an object during its life time.
  - Define a state machine to model the states of an object.

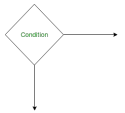
- Benefits of a State Diagram:
  - To model the reactive systems. Reactive systems can be defined as a system that responds to external or internal events.
  - To model the object states of a system.
  - To identify the events responsible for state changes.
  - Forward and reverse engineering.

Elements of State Diagram

a. Initial State:
   - It defines the initial state (beginning) of a system, and it is represented by a black filled circle.

b. Final State:
   - It represents the final state (end) of a system. It is denoted by a filled circle present within a circle.

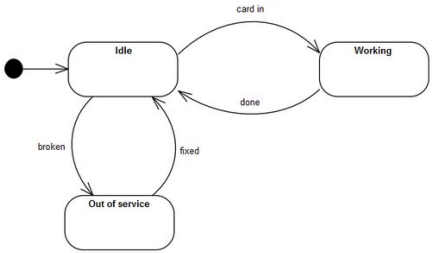
c. Decision Box:
   - It is of diamond shape that represents the decisions to be made on the basis of an evaluated guard.
   - It makes sure that the control flow or object flow will follow only one path.



Elements of State Diagram …. Contd.

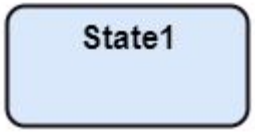
d. Transition:

- A change of control from one state to another due to the occurrence of some event is termed as a transition.
- It is represented by an arrow labeled with an event due to which the change has ensued.



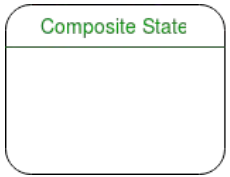
e. State/ State Box:

- It depicts the conditions or circumstances of a particular object of a class at a specific point of time.
- A rectangle with round corners is used to represent the state box.



Elements of State Diagram …. Contd.

f. Composite State:
   - We use a rounded rectangle to represent a composite state.
   - We represent a state with internal activities using a composite state.



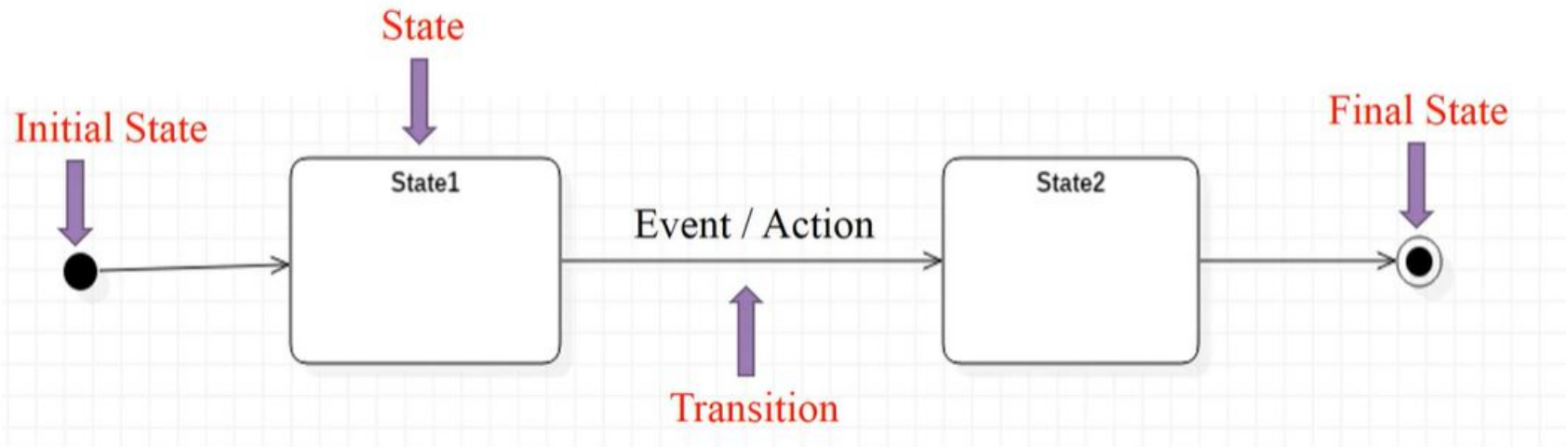
How to draw a State Diagram

- A state diagram is a graphic representation of a state machine.
- It shows a behavioral model consisting of states, transitions, and actions, as well as the events that affect these.

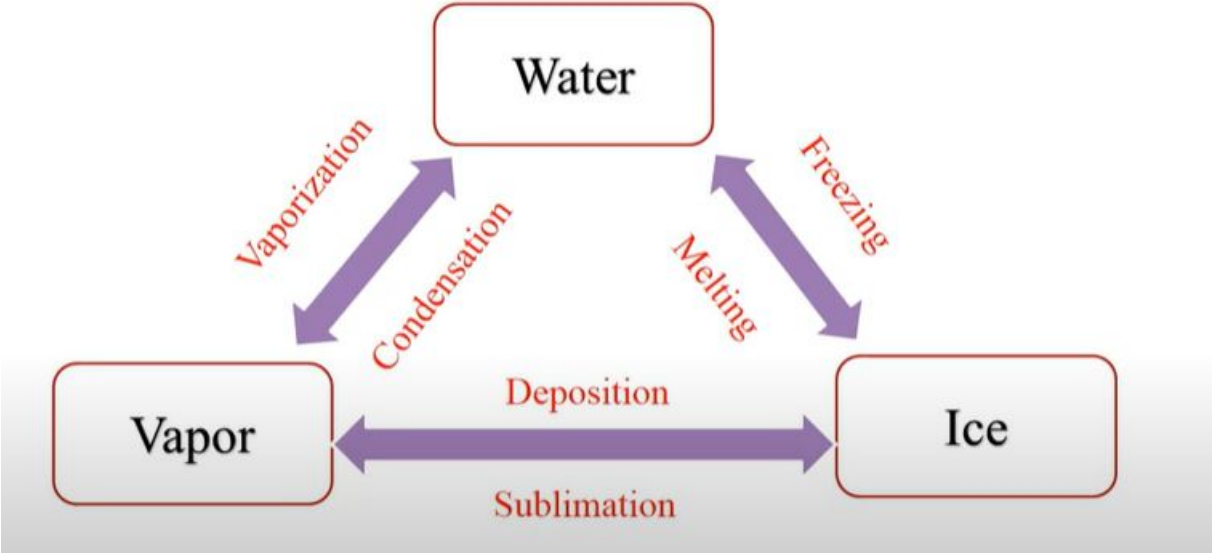
- Steps to draw a state diagram –
  - Identify the initial state and the final terminating states.
  - Identify the possible states in which the object can exist (boundary values corresponding to different attributes guide us in identifying different states).
  - Label the events which trigger these transitions.

  - https://www.youtube.com/watch?v=MRDyms57Uho

Example of State Diagram

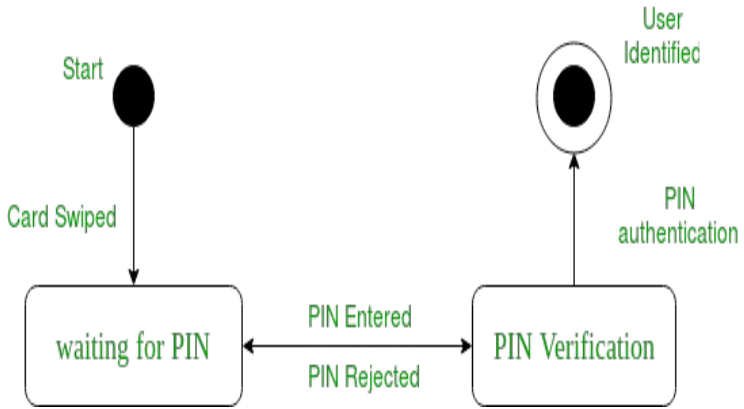


Example of State Diagram …. Contd.

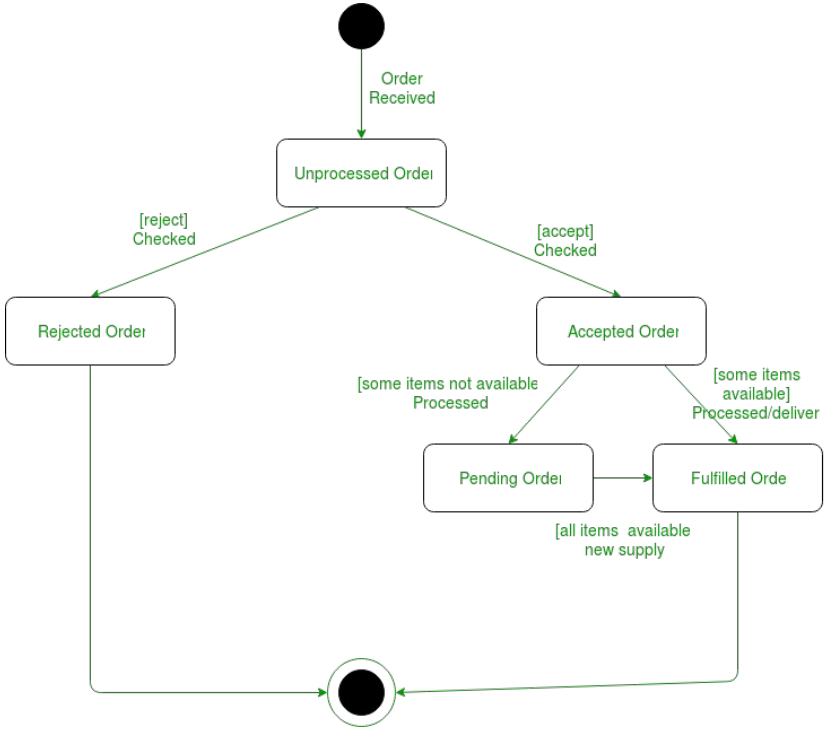


Example of State Diagram …. Contd.

Ex. User Verification

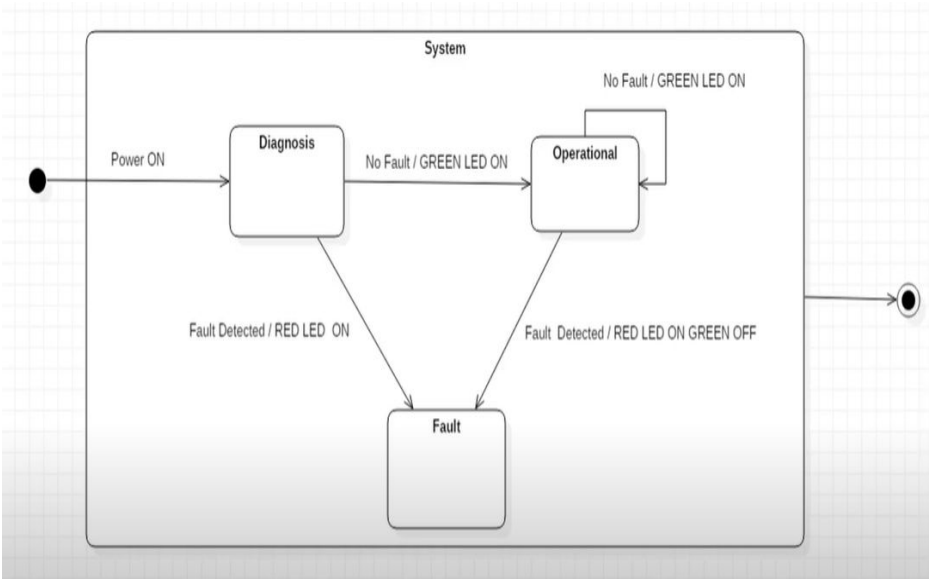


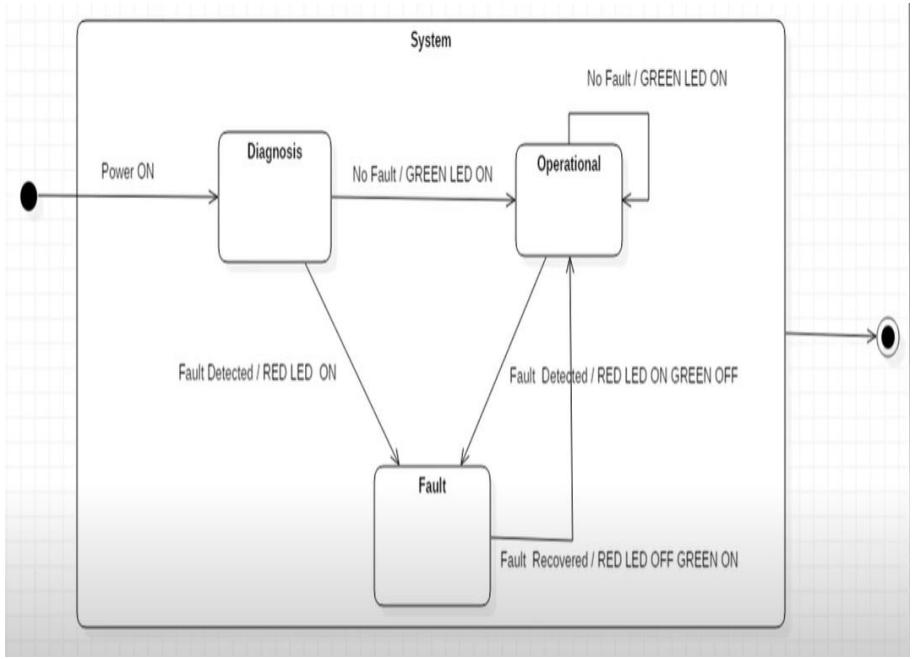
Online Order



Example of State Diagram …. Contd.

Ex. States of a System



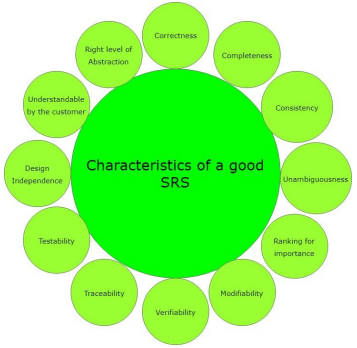


Software Requirement Specification (SRS)

- A software requirements specification (SRS) is a document that describes what the software will do and how it will be expected to perform.
- It also describes the functionality the product needs to fulfill all stakeholders (business, users) needs. A typical SRS includes: A purpose.
- The purpose of SRS document is to provide a detailed overview of our software product, its parameters and goals.
  - This document describes the project's target audience and its user interface, hardware and software requirements.
- An SRS minimizes the time and effort required by developers to achieve desired goals and also minimizes the development cost.

Good Qualities of SRS

- Following are the characteristics of a good SRS document:



i. Correctness:
   - User review is used to ensure the correctness of requirements stated in the SRS.
   - SRS is said to be correct if it covers all the requirements that are actually expected from the system.

ii. Completeness:
   - Completeness of SRS indicates every sense of completion including the numbering of all the pages, resolving the to be determined parts to as much extent as possible as well as covering all the functional and non-functional requirements properly.

Good Qualities of SRS …. Contd.

iii. Consistency:

- Requirements in SRS are said to be consistent if there are no conflicts between any set of requirements.
- Examples of conflict include differences in terminologies used at separate places, logical conflicts like time period of report generation, etc.

iv. Unambiguousness:

- A SRS is said to be unambiguous if all the requirements stated have only 1 interpretation.
- Some of the ways to prevent unambiguousness include the use of modelling techniques like ER diagrams, proper reviews and buddy checks, etc.

Good Qualities of SRS …. Contd.

v. Ranking for importance and stability:

- There should a criterion to classify the requirements as less or more important or more specifically as desirable or essential.
- An identifier mark can be used with every requirement to indicate its rank or stability.

vi. Modifiability:

- SRS should be made as modifiable as possible and should be capable of easily accepting changes to the system to some extent.
- Modifications should be properly indexed and cross-referenced.

vii. Verifiability:

- A SRS is verifiable if there exists a specific technique to quantifiably measure the extent to which every requirement is met by the system.
- For example, a requirement stating that the system must be user-friendly is not verifiable and listing such requirements should be avoided.

Good Qualities of SRS …. Contd.

viii. Traceability:

- One should be able to trace a requirement to design component and then to code segment in the program.
- Similarly, one should be able to trace a requirement to the corresponding test cases.

ix. Design Independence:

- There should be an option to choose from multiple design alternatives for the final system.
- More specifically, the SRS should not include any implementation details.

x. Testability:

- A SRS should be written in such a way that it is easy to generate test cases and test plans from the document.

Good Qualities of SRS …. Contd.

xi. Understandable by the customer:

- An end user maybe an expert in his/her specific domain but might not be an expert in computer science.
- Hence, the use of formal notations and symbols should be avoided to as much extent as possible.
- The language should be kept easy and clear.

xii. Right level of abstraction:

- If the SRS is written for the requirements phase, the details should be explained explicitly.
- Whereas, for a feasibility study, fewer details can be used. Hence, the level of abstraction varies according to the purpose of the SRS.

How to Write a SRS Document

A. 1. Create an Outline (Or Use an SRS Template)

- First step is to create an outline for your software requirements specification.
- It can be something you create yourself or use an existing SRS template.

The outline should look like:

1. Introduction
    - 1.1 Purpose
    - 1.2 Intended Audience
    - 1.3 Intended Use
    - 1.4 Scope
    - 1.5 Definitions and Acronyms
2. Overall Description
    - 2.1 User Needs
    - 2.2 Assumptions and Dependencies
3. System Features and Requirements
    - 3.1 Functional Requirements
    - 3.2 External Interface Requirements
    - 3.3 System Features
    - 3.4 Nonfunctional Requirements

Writing a SRS Document ….Contd.

B. Start With a Purpose

The introduction to SRS is very important. It sets the expectation for the product you are building. So, start by defining the purpose of your product.

Intended Audience and Intended Use

- Define who in your organization will have access to the SRS — and how they should use it.
- This may include developers, testers, and project managers.
- It could also include stakeholders in other departments, including leadership teams, sales, and marketing.

Product Scope

- Describe the software being specified.
- Include benefits, objectives, and goals.
- This should relate to overall business goals, especially if teams outside of development will have access to the SRS.

Definitions and Acronyms

- It is smart to include a risk definition.
- Avoiding risk is top-of-mind for many developers — especially those working on safety-critical development teams.

Writing a SRS Document ….Contd.

C. Give an Overview of What You will Build

Your next step is to give a description of what you are going to build. Is it an update to an existing product? Is it a new product? Is it an add-on to a product you have already created?

These are important to describe upfront, so everyone knows what you're building.
You should also describe why you are building it and who it is for.

User Needs

- User needs — or user classes and characteristics — are critical.
- Define who is going to use the product and how.
- Define primary and secondary users who will use the product on a regular basis.
- Also, define the needs of a separate buyer of the product (who may not be a primary/secondary user).

Assumptions and Dependencies

- There might be factors that impact your ability to fulfill the requirements outlined in your SRS. What are those factors?
- Are there any assumptions you are making with the SRS that could turn out to be false? You should include those here, as well.
- Finally, you should note if your project is dependent on any external factors. This might include software components you are reusing from another project.

Writing a SRS Document ….Contd.

D. Detail Your Specific Requirements

This section is key for the development team. This is where you detail the specific requirements for building your product.

i. Functional Requirements

- Functional requirements are essential to building your product.
- Within these functional requirements, you may have a subset of risks and requirements.

ii. External Interface Requirements

- External interface requirements are types of functional requirements.
- They are important for embedded systems.
- They outline how the product will interface with other components.

Writing a SRS Document ….Contd.

There are several types of interfaces you may have requirements for, including:

- User
- Hardware
- Software
- Communications

iii. System Features

- System features are types of functional requirements.
- These are features that are required in order for a system to function.

iv. Other Nonfunctional Requirements

- Nonfunctional requirements can be just as important as functional ones.
- These include:
  - Performance
  - Safety
  - Security
  - Quality

Writing a SRS Document ….Contd.

- The importance of this type of requirement may vary depending on your industry.
- Safety requirements, for example, will be critical in the medical device industry.

IEEE also provides guidance for writing software requirements specifications, if you are a member.

F. Get Approval for the SRS

- Once you have completed the SRS, you will need to get it approved by key stakeholders.
- And everyone should be reviewing the latest version of the document.