

# Software Engineering

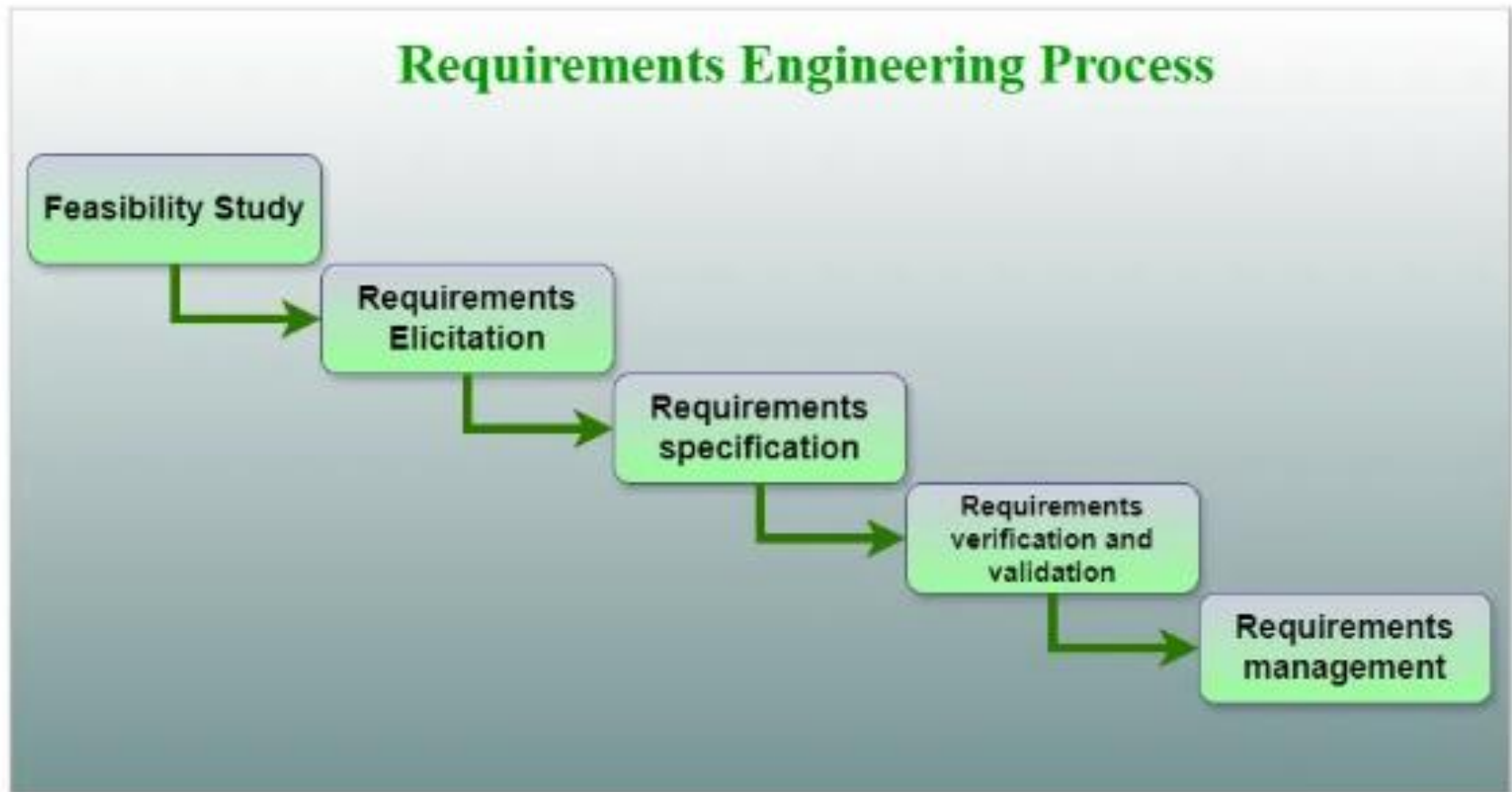
Code-2050302105

## Unit-2 Requirement Engineering

# Requirements Engineering

- A systematic and strict approach to the definition, creation, and verification of requirements for a software system is known as requirements engineering. To guarantee the effective creation of a software product, the requirements engineering process entails several tasks that help in understanding, recording, and managing the demands of stakeholders.

# Process



*Requirements Engineering Process*

# Advantages

- Helps ensure that the software being developed meets the needs and expectations of the stakeholders
- Can help identify potential issues or problems early in the development process, allowing for adjustments to be made before significant
- Helps ensure that the software is developed in a cost-effective and efficient manner
- Can improve communication and collaboration between the development team and stakeholders
- Helps to ensure that the software system meets the needs of all stakeholders.

# Dis-Advantages

- Can be time-consuming and costly, particularly if the requirements-gathering process is not well-managed
- Can be difficult to ensure that all stakeholders' needs and expectations are taken into account
- It Can be challenging to ensure that the requirements are clear, consistent, and complete
- Changes in requirements can lead to delays and increased costs in the development process.
- As a best practice, Requirements engineering should be flexible, adaptable, and should be aligned with the overall project goals.

# Types

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

# Functional Requirement

- Search option given to user to search from various invoices.
- User should be able to mail any report to management.
- Users can be divided into groups and groups can be given separate rights.
- Should comply business rules and administrative functions.
- Software is developed keeping downward compatibility intact.

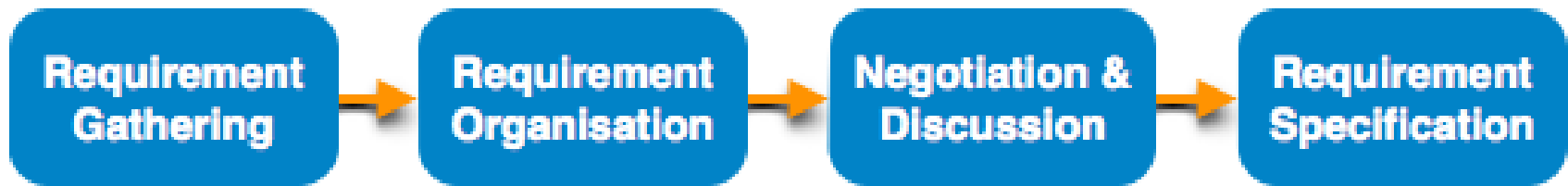
# Non-functional Requirement

- Security
- Logging
- Storage
- Configuration
- Performance
- Cost
- Interoperability
- Flexibility
- Disaster recovery
- Accessibility



# Requirement Elicitation Process

- **Requirements elicitation** is the process of gathering and defining the requirements for a software system. The goal of requirements elicitation is to ensure that the software development process is based on a clear and comprehensive understanding of the customer's needs and requirements.



# Conti..

- **Requirements gathering** - The developers discuss with the client and end users and know their expectations from the software.
- **Organizing Requirements** - The developers prioritize and arrange the requirements in order of importance, urgency and convenience.
- **Negotiation & discussion** - If requirements are ambiguous or there are some conflicts in requirements of various stakeholders, if they are, it is then negotiated and discussed with stakeholders. Requirements may then be prioritized and reasonably compromised.
- The requirements come from various stakeholders. To remove the ambiguity and conflicts, they are discussed for clarity and correctness. Unrealistic requirements are compromised reasonably.
- **Documentation** - All formal & informal, functional and non-functional requirements are documented and made available for next phase processing.

# Importance of Requirements Elicitation

- **Compliance with Business Objectives:** The process of elicitation guarantees that the software development endeavors are in harmony with the wider company aims and objectives. Comprehending the business context facilitates the development of a solution that adds value for the company.
- **User Satisfaction:** It is easier to create software that fulfills end users' needs and expectations when they are involved in the requirements elicitation process. Higher user pleasure and acceptance of the finished product are the results of this.
- **Time and Money Savings:** Having precise and well-defined specifications aids in preventing miscommunication and rework during the development phase. As a result, there will be cost savings and the project will be completed on time.
- **Compliance and Regulation Requirements:** Requirements elicitation is crucial for projects in regulated industries to guarantee that the software conforms with applicable laws and norms. In industries like healthcare, finance, and aerospace, this is crucial.
- **Traceability and Documentation:** Throughout the [software development process](#), traceability is based on well-documented requirements. Traceability helps with testing, validation, and maintenance by ensuring that every part of the software can be linked to a particular requirement.

# Adv. of Requirements Elicitation

- **Clear requirements:** Helps to clarify and refine customer requirements.
- **Improves communication:** Improves communication and collaboration between stakeholders.
- **Results in good quality software:** Increases the chances of developing a software system that meets customer needs.
- **Avoids misunderstandings:** Avoids misunderstandings and helps to manage expectations.
- **Supports the identification of potential risks:** Supports the identification of potential risks and problems early in the development cycle.
- **Facilitates development of accurate plan:** Facilitates the development of a comprehensive and accurate project plan.
- **Increases user confidence:** Increases user and stakeholder confidence in the software development process.
- **Supports identification of new business opportunities:** Supports the identification of new business opportunities and revenue streams.

# Dis-adv. of Requirements Elicitation

- **Time-consuming:** It can be time-consuming and expensive.
- **Skills required:** Requires specialized skills and expertise.
- **Impacted by changing requirements:** This may be impacted by changing business needs and requirements.
- **Impacted by other factors:** Can be impacted by political and organizational factors.
- **Lack of commitment from stakeholders:** This can result in a lack of buy-in and commitment from stakeholders.
- **Impacted by conflicting priorities:** Can be impacted by conflicting priorities and competing interests.
- **Sometimes inaccurate requirements:** This may result in incomplete or inaccurate requirements if not properly managed.
- **Increased development cost:** This can lead to increased development costs and decreased efficiency if requirements are not well-defined.

# Use cases

- The Use-case model is defined as a model which is used to show how users interact with the system in order to solve a problem. As such, the use case model defines the user's objective, the interactions between the system and the user, and the system's behavior required to meet these objectives.
- We use a use-case diagram to graphically portray a subset of the model in order to make the communication simpler.

# Conti..

- Packages may include a use-case model, which is used to organize the model to simplify the analysis, planning, navigation, communication, development and maintenance.
- The use-case model acts as an integrated thread in the development of the entire system. The use-case model is used like the main specification of the system functional requirements as the basis for design and analysis, as the basis for user documentation, as the basis of defining test cases, and as an input to iteration planning.

# Conti..

- Ivar Jacobson, in the year 1986, originally formulated textual and visual modeling methods to specify use cases.
- And in the year 1992, his co-authored book named Object-Oriented Software Engineering - A Use Case Driven Approach, assisted with promoting the strategy for catching functional requirements, particularly in software development.



# Basic Model

- **Actor:** Usually, actors are people involved with the system defined on the basis of their roles. An actor can be anything such as human or another external system.
- **Use Case:** The use case defines how actors use a system to accomplish a specific objective. The use cases are generally introduced by the user to meet the objectives of the activities and variants involved in the achievement of the goal.

# Basic Model

- **Associations:** Associations are another component of the basic model. It is used to define the associations among actors and use cases they contribute in. This association is called communicates-association.
- **boardmix**

# Model Components

- **Subject:** The subject component is used to represent the boundary of the system of interest.
- **Use-Case Package:** We use the model component in order to structure the use case model to make simpler the analysis, planning, navigation, and communication. Suppose there are various actors or use cases. In that case, we can also use use-case packages in order to further structure the use-case model in much the similar way we use directories or folders to organize the information on our hard-disk.
- For various reasons, we divide the use-case model into the use-case packages, containing:
  - To help parallel development by partitioning the problem into bite-sized parts.
  - To improve communication with various stakeholders by making packaging containing actors, use cases and related to the specific stakeholder.

# Model Components

- **Generalizations:** Generalizations mean the association between the actors in order to help re-use of common properties.
- **Dependencies:** In UML, various types of dependencies are defined between use cases. In particular, <<include>> and <<extend>>.
- We use <<include>> dependency to comprise shared behavior from an included use case into a base use case to use common behavior.
- We use <<extend>> dependency to include optional behavior from an extended use-case into an extended use case.

# Use case diagram used for

- Examining the system's requirements.
- Capturing the system's Functionalities.
- We use use-case diagram in order to modeling the general idea behind the system.
- System's Forward and reverse engineering using several test cases.
- Complex visual designing of software.

# Importance of Use case diagram

- Use-case diagram provides an outline related to all components in the system. Use-case diagram helps to define the role of administrators, users, etc.
- The use-Case diagram helps to provide solutions and answers to various questions that may pop up if you begin a project unplanned.
- It helps us to define the needs of the users extensively and explore how it will work.

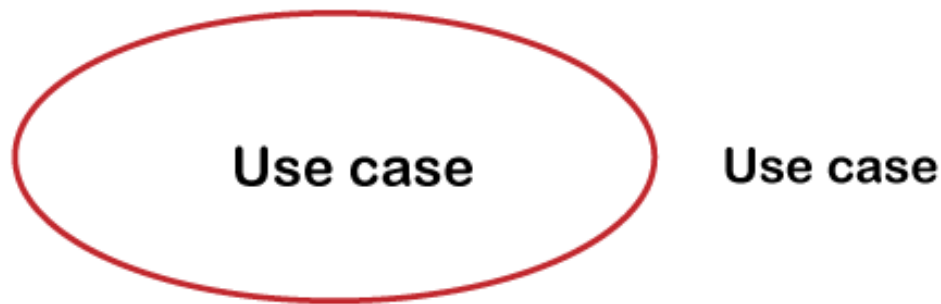
# Basic use case diagram symbols

- **Systems:**
- With the help of the rectangle, we can draw the boundaries of the system, which includes use-cases. We need to put the actors outside the system's boundaries.



# Conti..

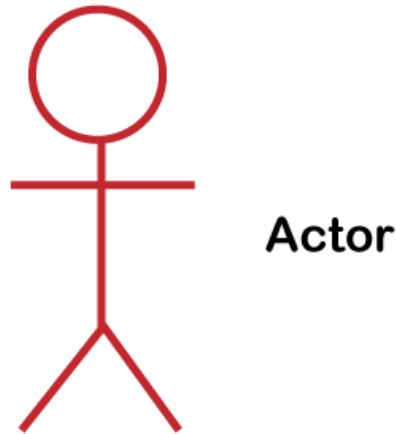
- Use case:
- With the help of the Ovals, we can draw the use-cases. With the verb we have to label the ovals in order to represent the functions of the system.





# Conti..

- **Actors:**
- Actors mean the system's users. If one system is the actor of the other system, then with the actor stereotype, we have to tag the actor system.



# Conti..

- Relationship:
- With the simple line we can represent relationships between an actor and use cases. For relationships between use-case, we use arrows which are labeled either "extends" or "uses". The "extends" relationship shows the alternative options under the specific use case. The "uses" relationship shows that single use-case is required to accomplish a job.

# Conti..

- Relationship:



Relationships



# Guidelines for Use-case

- **Actors**

- **The actor's name should be meaningful and relevant to the business**

If the use-case is interacting with the outside organization, then we have to give the actor's name with the function instead of the organization name, such as Airline company is better than the PanAir).

- **Place inheriting actors below the parent actor**

We have to place the inheriting actors below the parent actor because it makes the actors more readable and easily highlights the use-cases, which are exact for that actor.

- **External Systems are actors**

If send-email is our use-case and when the use-case interrelates with the email management software, then in this case, the software is an actor to that specific user-case.

# Conti..

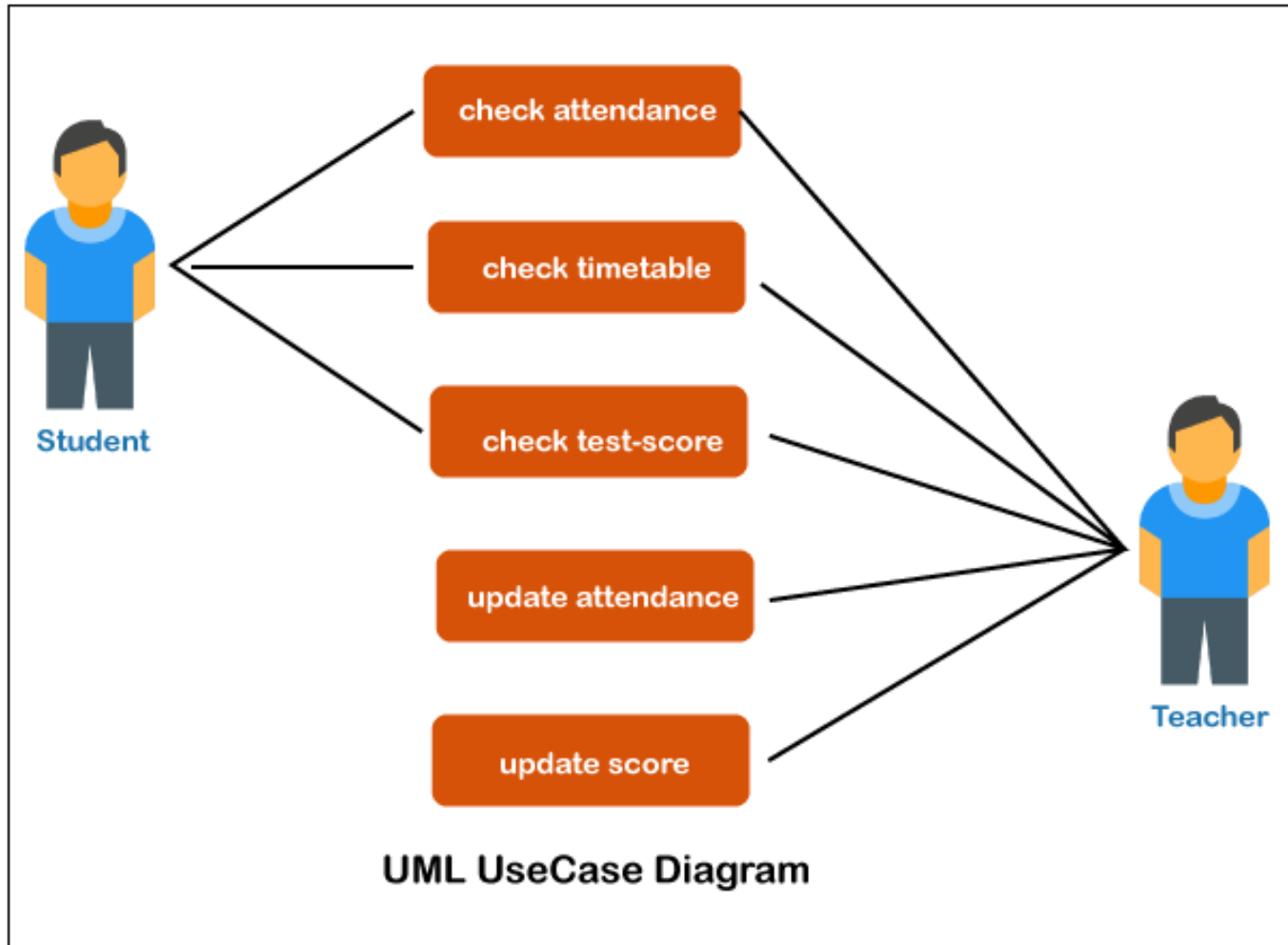
- **Use-Cases**
- **The name of the use-case begins with a verb**  
The use-case models action, so the name of the use-case must start with a verb.
- **The name of the use-case must be descriptive**  
The use-case is created to provide more information to others who are looking at a diagram, such as instead of "Print," "print Invoice is good.
- **Put the use-cases to the right of the included use-cases.**  
In order to add clarity and enhance readability, we have to place the included use-cases to the right of the invoking use-cases.
- **Place inheriting use-case below the parent use-case**  
In order to enhance the diagram's readability, we have to place the inheriting use-case below the parent use-case.

# Conti..

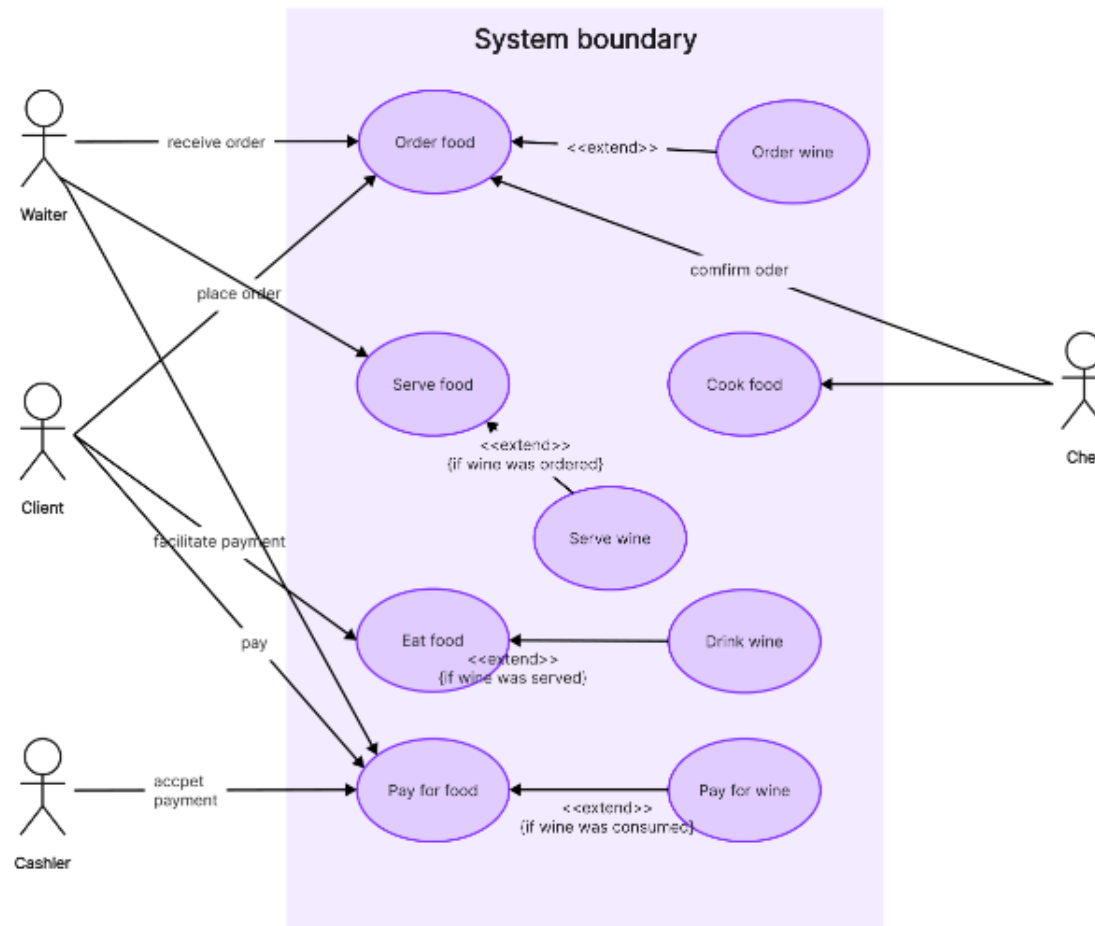
- **Relationships**

- When we are using <<extend>> arrow, points to the base use-case.
- When we are using <<include>> then arrow points to the comprised use-case.
- Actor and use-case relationship do not display arrows.
- <<extend>> may have an optional extension condition.
- <<include>> and <<extend>> both are shown as dashed arrows.

# Example



# Example





# Requirement Model

- The analysis model's goal is to provide a description of the informational, functional, and behavioural domains required for a computer-based system. The model evolves dynamically as you learn more about the system to be developed and other stakeholders gain a better understanding of what they actually require.

# Elements of Requirement Model

- **Scenario-based elements:** A scenario-based approach is used to describe the system from the perspective of the user. For example, basic use cases and their related use-case diagrams, evolve into more complicated template-based use cases. Scenario-based requirements model elements are frequently the first parts of the model to be produced. There are three levels of elaboration shown, ending in a scenario-based portrayal.

# Conti..

- **Class-based element:** Each usage scenario entails a collection of objects that are modified as an actor interacts with the system. These objects are classified as classes— a collection of things with similar characteristics and behaviour.
- **Behavioural elements:** The behaviour of a computer-based system can have a significant impact on the design and implementation techniques used. As a result, the requirements model must include modelling elements that represent behaviour. The state diagram is one approach of expressing a system's behaviour by illustrating its states and the events that cause the system to change state. Any externally observable form of conduct is referred to as a state. Furthermore, the state diagram indicates activities taken as a result of a specific event.

# Conti..

- **Flow-oriented elements:** As data moves through a computer-based system, it is transformed. The system accepts input in a variety of formats, transforms it using functions, and produces output in a variety of forms. A control signal transmitted by a transducer, a series of numbers written by a human operator, a packet of information transmitted over a network link, or a large data file retrieved from secondary storage can all be used as input. The transform(s) may consist of a single logical comparison, a complex numerical algorithm, or an expert system's rule-inference approach.

# Negotiating Requirements

- The inception, elicitation, and elaboration tasks in an ideal requirements engineering setting determine customer requirements in sufficient depth to proceed to later software engineering activities. You might have to negotiate with one or more stakeholders. Most of the time, stakeholders are expected to balance functionality, performance, and other product or system attributes against cost and time-to-market.
- The goal of this discussion is to create a project plan that meets the objectives of stakeholders while also reflecting the real-world restrictions (e.g., time, personnel, and budget) imposed on the software team. The successful negotiations aim for a “win-win” outcome. That is, stakeholders benefit from a system or product that meets the majority of their needs, while you benefit from working within realistic and reasonable budgets and schedules.

# Conti..

- Identifying the major stakeholders in the system or subsystem.
- Establishing the stakeholders' "win conditions."
- Negotiation of the win conditions of the stakeholders in order to reconcile them into a set of win-win conditions for all people involved.
- Risk Management
- Iterative Negotiation
- Techniques for Effective Negotiation
- Tools for Requirement Negotiation

# Validating Requirements

- **Requirements validation** is the process of checking that requirements defined for development, define the system that the customer wants. To check issues related to requirements, we perform requirements validation. We typically use requirements validation to check errors at the initial phase of development as the error may increase excessive rework when detected later in the development process. In the requirements validation process, we perform a different type of test to check the requirements mentioned in the [Software Requirements Specification \(SRS\)](#), these checks include:

# Conti..

- **Completeness checks**
- **Consistency checks**
- **Validity checks**
- **Realism checks**
- **Ambiguity checks**
- **Variability**



# Requirement

## 1. Test Case Generation

The requirement mentioned in the SRS document should be testable, the conducted tests reveal the error present in the requirement. It is generally believed that if the test is difficult or impossible to design, this usually means that the requirement will be difficult to implement and it should be reconsidered.

# Requirement

## 2. Prototyping

In this validation technique the prototype of the system is presented before the end-user or customer, they experiment with the presented model and check if it meets their need. This type of model is mostly used to collect feedback about the requirement of the user.

# Requirement

## 3. Requirements Reviews

In this approach, the SRS is carefully reviewed by a group of people including people from both the contractor organizations and the client side, the reviewer systematically analyses the document to check errors and ambiguity.

# Requirement

## 4. Automated Consistency Analysis

This approach is used for the automatic detection of an error, such as non-determinism, missing cases, a type error, and circular definitions, in requirements specifications. First, the requirement is structured in formal notation then the CASE tool is used to check the in-consistency of the system, The report of all inconsistencies is identified, and corrective actions are taken.

# Requirement

## 5. Walk-through

- A walkthrough does not have a formally defined procedure and does not require a differentiated role assignment.
- Checking early whether the idea is feasible or not.
- Obtaining the opinions and suggestions of other people.
- Checking the approval of others and reaching an agreement.

# Requirement

## 6. Simulation

- Simulating system behavior in order to verify requirements is known as simulation. This method works especially well for complicated systems when it is possible to replicate real-world settings and make sure the criteria fulfil the desired goals.

# Requirement

- **7. Checklists for Validation**
- It employs pre-made checklists to methodically confirm that every prerequisite satisfies predetermined standards. Aspects like completeness, clarity and viability can all be covered by checklists.

# Importance

- **Accuracy and Clarity:** It makes sure that the requirements are precise, unambiguous and clear. This helps to avoid miscommunications and misunderstandings that may result in mistakes and more effort in subsequent phases of the project.
- **User Satisfaction:** It confirms that the requirements meet the wants and expectations of the users, which helps to increase user happiness. This aids in providing a product that satisfies consumer needs and improves user experience as a whole.



# Conti..

- **Early Issue Identification:** It makes it easier to find problems, ambiguities or conflicts in the requirements early on. It is more economical to address these issues early in the development phase rather than later, when the project is far along.
- **Prevents the Scope Creep:** It ensures that the established requirements are well stated and recorded, which helps to prevent scope creep. By establishing defined parameters for the project's scope, requirements validation helps to lower the possibility of uncontrollably changing course.

## Conti..

- **Improving Quality:** It enhances the software product's overall quality. By detecting and resolving possible quality problems early in the development life cycle, requirements validation contributes to the creation of a more durable and dependable final product.

# Advantages

- **Improved quality of the final product:** By identifying and addressing requirements early on in the development process, using validation techniques can improve the overall quality of the final product.
- **Reduced development time and cost:** By identifying and addressing requirements early on in the development process, using validation techniques can reduce the likelihood of costly rework later on.

# Advantages

- **Increased user involvement:** Involving users in the validation process can lead to increased user buy-in and engagement in the project.
- **Improved communication:** Using validation techniques can improve communication between stakeholders and developers, by providing a clear and visual representation of the software requirements.

# Advantages

- **Easy testing and validation:** A prototype can be easily tested and validated, allowing stakeholders to see how the final product will work and identify any issues early on in the development process.
- **Increased alignment with business goals:** Using validation techniques can help to ensure that the requirements align with the overall business goals and objectives of the organization.

# Advantages

- **Traceability:** This technique can help to ensure that the requirements are being met and that any changes are tracked and managed.
- **Agile methodologies:** Agile methodologies provide an iterative approach to validate requirements by delivering small chunks of functionality and getting feedback from the customer.

# Dis-advantages

- **Increased time and cost:** Using validation techniques can be time-consuming and costly, especially when involving multiple stakeholders.
- **Risk of conflicting requirements:** Using validation techniques can lead to conflicting requirements, which can make it difficult to prioritize and implement the requirements.
- **Risk of changing requirements:** Requirements may change over time and it can be difficult to keep up with the changes and ensure that the project is aligned with the updated requirements.

# Dis-advantages

- **Mis-interpretation and mis-communication:** Misinterpretation and miscommunication can occur when trying to understand the requirements.
- **Dependence on the tool:** The team should be well-trained on the tool and its features to avoid dependency on the tool and not on the requirement.
- **Limited validation:** The validation techniques can only check the requirement that is captured and may not identify the requirement that is missed



# Dis-advantages

- **Limited to functional requirements:** Some validation techniques are limited to functional requirements and may not validate non-functional requirements.

# Modeling Strategies

1. Flow Oriented Modeling
2. Class-based Modeling

# Flow-Oriented Model

- It shows how data objects are transformed by processing the function.
- i. Data flow model:** It is a graphical technique. It is used to represent information flow.
- The data objects are flowing within the software and transformed by processing the elements.
- The data objects are represented by labeled arrows. Transformation are represented by circles called as bubbles.
- DFD shown in a hierarchical fashion. The DFD is split into different levels. It also called as 'context level diagram'.

# Conti..

**ii. Control flow model:** Large class applications require a control flow modeling.

- The application creates control information instated of reports or displays.
- The applications process the information in specified time.
- An event is implemented as a boolean value.  
**For example,** the boolean values are true or false, on or off, 1 or 0.

# Conti..

**iii. Control Specification:** A short term for control specification is CSPEC.

- It represents the behaviour of the system.
- The state diagram in CSPEC is a sequential specification of the behaviour.
- The state diagram includes states, transitions, events and activities.
- State diagram shows the transition from one state to another state if a particular event has occurred.

# Conti..

**iv. Process Specification:** A short term for process specification is PSPEC.

- The process specification is used to describe all flow model processes.
- The content of process specification consists narrative text, Program Design Language(PDL) of the process algorithm, mathematical equations, tables or UML activity diagram.

# Class Based Model

- Class based modeling represents the object. The system manipulates the operations.
- The elements of the class based model consist of classes and object, attributes, operations, class – responsibility - collaborator (CRS) models.

# Conti..

- **Classes**

Classes are determined using underlining each noun or noun clause and enter it into the simple table.

**Classes are found in following forms:**

- **External entities:** The system, people or the device generates the information that is used by the computer based system.
- **Things:** The reports, displays, letter, signal are the part of the information domain or the problem.
- **Occurrences or events:** A property transfer or the completion of a series or robot movements occurs in the context of the system operation.
- **Roles:** The people like manager, engineer, salesperson are interacting with the system.
- **Organizational units:** The division, group, team are suitable for an application.
- **Places:** The manufacturing floor or loading dock from the context of the problem and the overall function of the system.
- **Structures:** The sensors, computers are defined a class of objects or related classes of objects.



# Conti..

- **Responsibility:** The key functions or duties that the class is responsible for.
- **Collaborator:** Other classes that the current class interacts with to fulfill its responsibilities.

# Behavioral Modeling

- Behavioral model describe the overall behavior of the system. There are two types of behavioral models that are used to describe the system behavior, one is data processing model and another is state machine models. Data processing models are also known as DFD (Data Flow Diagram) which is used to show how data is processed as it moves through the system. State machine model is also known as State diagram which is used to show how the system will react with external events.

# Conti..

## **Data Flow Diagram:**

- Data flow diagram is used to model the system's data processing.
- It is also called as the Functional model as it is a graphical representation of an enterprise function within a defined scope. Data flow diagram shows end to end data processing.
- It can be easily converted into software as they just represent flow of the data objects. DFD diagram enable Software engineer to develop a model of the information domain and Functional domain at the same time.
- It provide a logical model of the system and show the flow of the data and the flow of logic involved.

# Conti..

## **Guidelines for Data Flow Diagram:**

- Level 0 DFD should depict software system as single bubble.
- Primary input and output are carefully noted.
- Refinement should begin by isolating candidate processes, data objects, data stores to be represented at the next level.
- All arrows and bubbles should be labeled with full names.
- Information flow continuity must be maintained from level to level. That means the data objects that flow into the system of any transformation at one level must be the same. Data objects that flow into transformation at the more refined level.
- One bubble at a time should be refined.

# Conti..



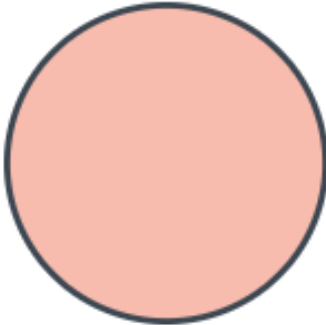
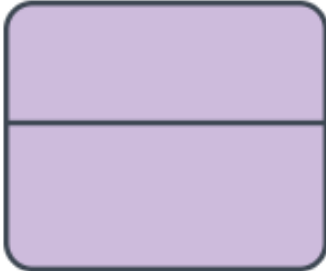




- Data flow diagrams were popularized in the late 1970s, arising from the book *Structured Design*, by computing pioneers Ed Yourdon and Larry Constantine. They based it on the “data flow graph” computation models by David Martin and Gerald Estrin. The structured design concept took off in the software engineering field, and the DFD method took off with it. It became more popular in business circles, as it was applied to business analysis, than in academic circles.
- Also contributing were two related concepts:
- Object Oriented Analysis and Design (OOAD), put forth by Yourdon and Peter Coad to analyze and design an application or system.
- Structured Systems Analysis and Design Method (SSADM), a waterfall method to analyze and design information systems. This rigorous documentation approach contrasts with modern agile approaches such as Scrum and Dynamic Systems Development Method (DSDM.)

# Conti..

## Notations are as follows:

- **Data Flow:** It represents the movement of data flow from a specific origin to a destination.
- **Process:** It represents the users, procedures, or devices that use the data.
- **Entity:** It represents the source of data or destination data external sources or destination of data which may be users, programs, organizations or other entities that interact with the system but are outside its boundary.
- **Data Store:** There can be a single DFD diagram or can be exploded into various levels like level 1, level 2, level 3, etc.

# Conti..

Notation	Yourdon and Coad	Gane and Sarson
External Entity		
Process		
Data Store		
Data Flow		

# Conti..

## DFD rules :

- Each process should have at least one input and an output.
- Each data store should have at least one data flow in and one data flow out.
- Data stored in a system must go through a process.
- All processes in a DFD go to another process or a data store.
- Example: LucidChart, Visual-Paradigm



# Conti..

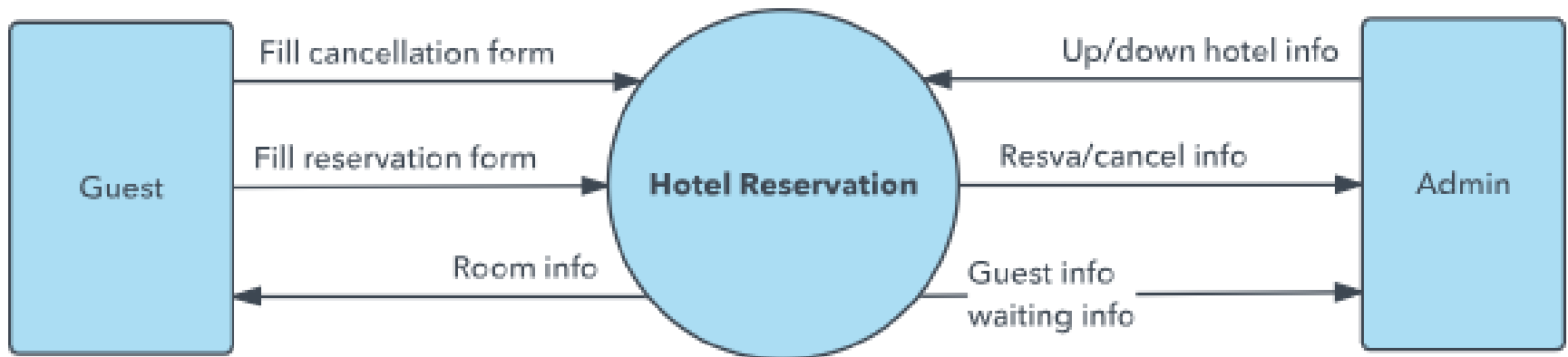
## **Levels in Data Flow Diagram (DFD)**

- **Level 0 DFD**
- **Level 1 DFD**
- **Level 2 DFD**

# Conti..

- **Level 0** is also called a Context Diagram. It's a basic overview of the whole system or process being analyzed or modeled. It's designed to be an at-a-glance view, showing the system as a single high-level process, with its relationship to external entities. It should be easily understood by a wide audience, including stakeholders, business analysts, data analysts and developers.

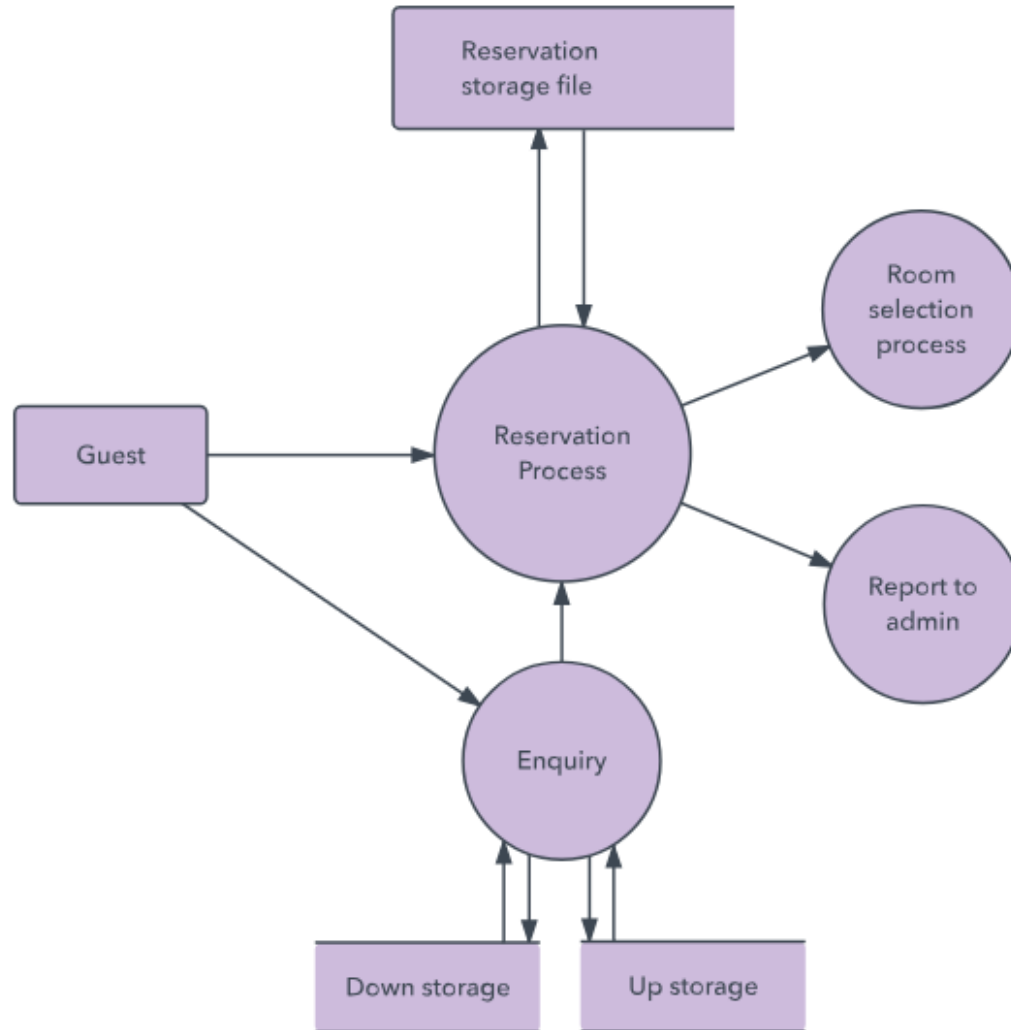
# Conti..



# Conti..

- **Level 1** provides a more detailed breakout of pieces of the Context Level Diagram. You will highlight the main functions carried out by the system, as you break down the high-level process of the Context Diagram into its subprocesses.

# Conti..



# Conti..

- **Level 2** then goes one step deeper into parts of Level 1. It may require more text to reach the necessary level of detail about the system's functioning

# Conti..



# State Diagram

- State diagram is a dynamic model which represents the changes of state that an object goes through during the lifetime in response to events. It is used to help the developer better understand any complex functionality of specialized areas of the system.

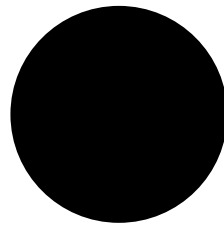


# Conti..

**Notations are as follows:**

## **1. Initial State**

It represents the start point of the diagram. It is also called as pseudo state where state has no variables and no activities.

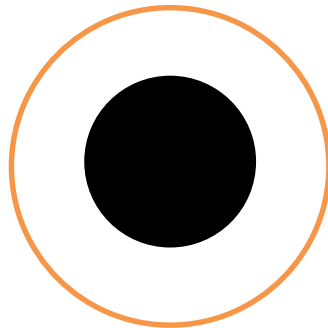


# Conti..

**Notations are as follows:**

## **2. Final State**

It represents the end point of the diagram. It is also pseudo state as it doesn't have any variable or activities. State diagram can have zero or more final states.



# Conti..

**Notations are as follows:**

## **3. State**

It represents the state of the object at an instant of time. State is a recognizable situation and exists over an interval of time.



# Conti..

**Notations are as follows:**

## **4. Transition**

It represents the changes from one state to other. It is denoted by an arrow. The event or action causing the transition is written beside the arrow, separated by slash. Trigger less transition occurs when the state completed an activity.

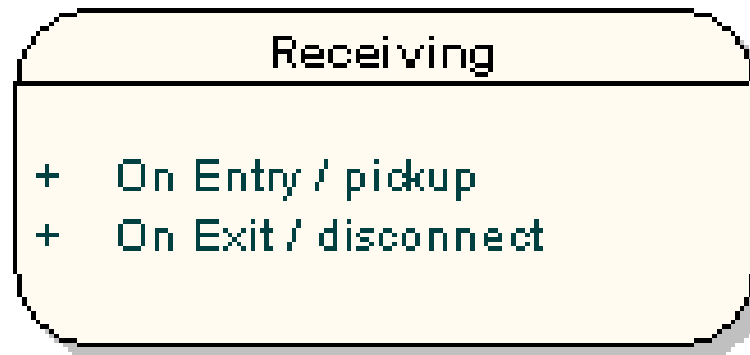


# Conti..

**Notations are as follows:**

## **5. Event and Action**

Trigger that causes a transition to occur and changes the state is called event or action.



# Conti..

**Notations are as follows:**

## **6. History State**

A flow may require that the object go into a wait state and on the occurrence of a certain event, go back to the state it was in, in this situation this notation is used.

## **7. Signal**

When even causes the trigger to be sent to a state that causes the transition, then that message sent by the event is called a signal.

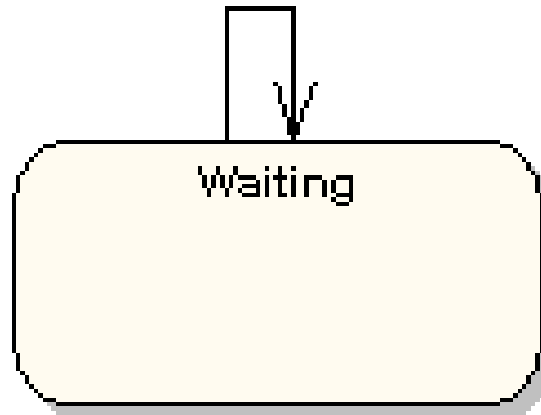
# Conti..

**Notations are as follows:**

## **8. Self Transition**

A state that has a transition that returns to itself is called self-transition.

after 2 seconds / poll input

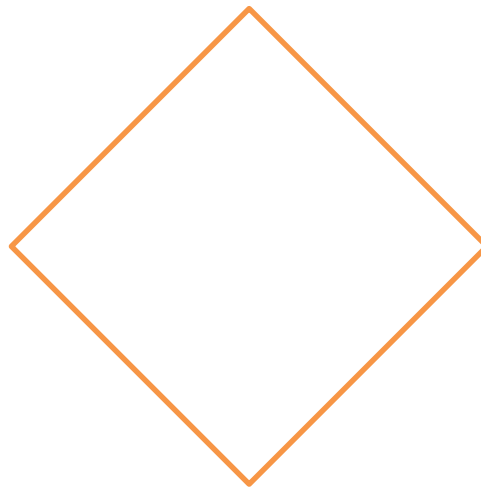


# Conti..

**Notations are as follows:**

## **8. Decision Box**

It is of diamond shape that represents the decisions to be made on the basis of an evaluated guard.





# Conti..

