Software Engineering-BSCE-301L

Module 3:

Modeling Requirements

Dr . Saurabh Agrawal

Faculty Id: 20165

School of Computer Science and Engineering

VIT, Vellore-632014

Tamil Nadu, India

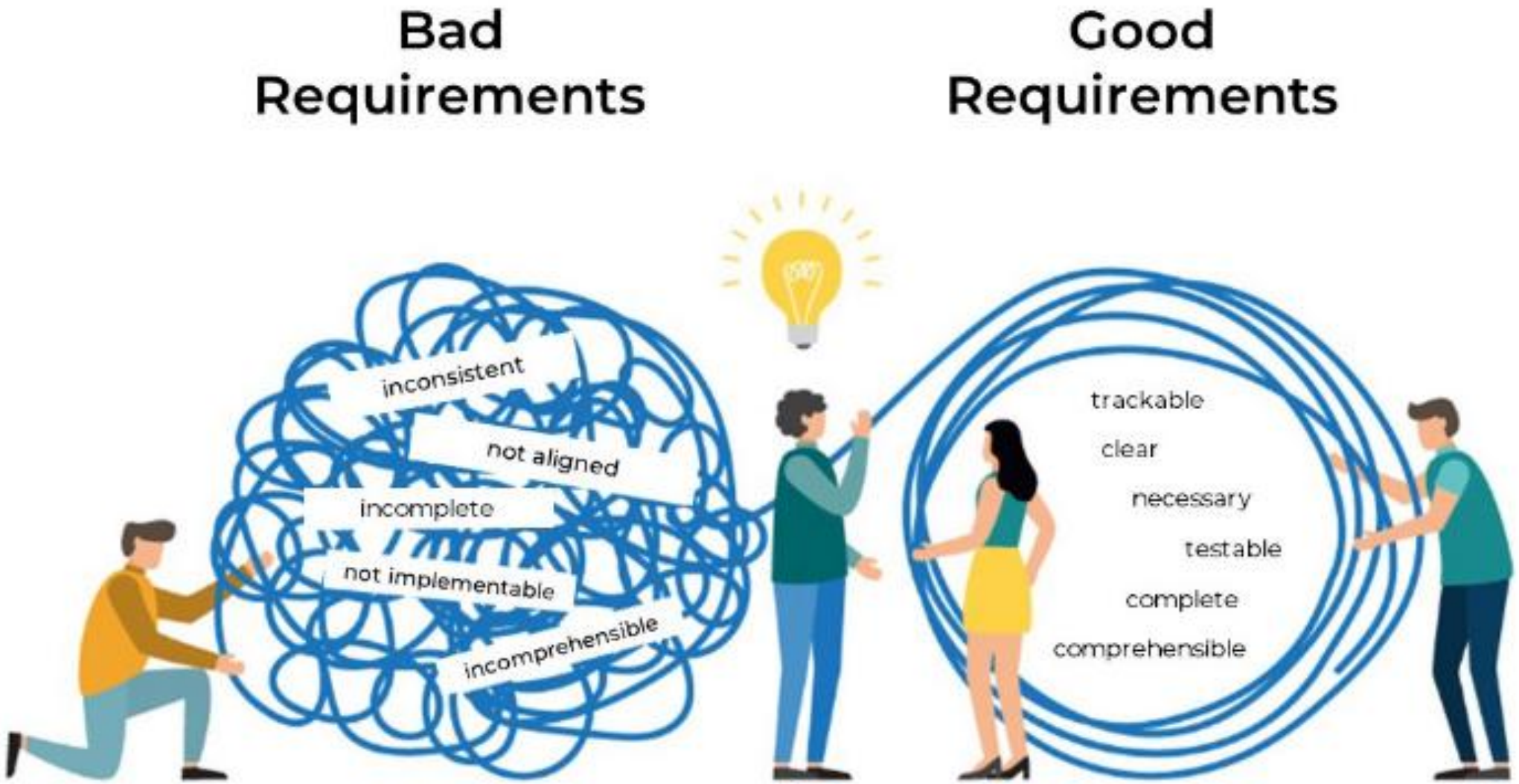
Outline

- Software Requirements and its Types
- Requirements Engineering Process
- Requirement Elicitation
- Requirements Elicitation techniques
- System Modeling – Requirements Specification and Requirement Validation
- Requirements management in Agile

Software Requirements and its Types

- The requirement can be defined as a high-level abstract statement or a detailed mathematical functional specification of a system's services, functions, and constraints.
- They are depictions of the characteristics and functionalities of the target system.
- Requirements denote the expectations of users from the software product.
- The requirement should be open to interpretation and detailed enough to understand.
- It is essential to know about software requirements because it minimizes the developer's time and effort and the development cost.
- We have also discussed requirement engineering and the process in one of our articles.

Software Requirements and its Types



Types of software requirements

Business requirements	User requirements	Software requirements
Outline measurable goals for the business.	Reflect specific user needs or expectations.	Identify features, functions, non-functional requirements and use cases.
Define the *why* behind a software project.	Describe the *who* of a software project.	Delve into the *how* of a software project.
Match project goals to stakeholder goals.	Highlight how users interact with it.	Describe software as functional modules and non-functional attributes.
Maintain a BRD with requirements, updates or changes.	Create a URS, or make them part of the BRD.	Compose an SRS, and, optionally, an FRS.

Software Requirements and its Types

❑Business requirements

❑Business needs drive many software projects.

❑A business requirements document (BRD) outlines measurable project goals for the business, users and other stakeholders.

❑Business analysts, leaders and other project sponsors create the BRD at the start of the project. This document defines the *why* behind the build.

❑For software development contractors, the BRD also serves as the basis for more detailed document preparation with clients.

❑A BRD is composed of one or more statements. No universally established format exists for BRD statements, but one common approach is to align goals:

❑ Write statements that match a project goal to a measurable stakeholder or business goal. The basic format of a BRD statement looks like:

"The [project name] software will [meet a business goal] in order to [realize a business benefit]."

Software Requirements and its Types

❑User Requirements

❑User requirements reflect the specific needs or expectations of the software's customers.

❑Organizations sometimes incorporate these requirements into a BRD, but an application that poses extensive user functionality or complex UI issues might justify a separate document specific to the needs of the intended user.

❑User requirements -- much like user stories -- highlight the ways in which customers interact with software.

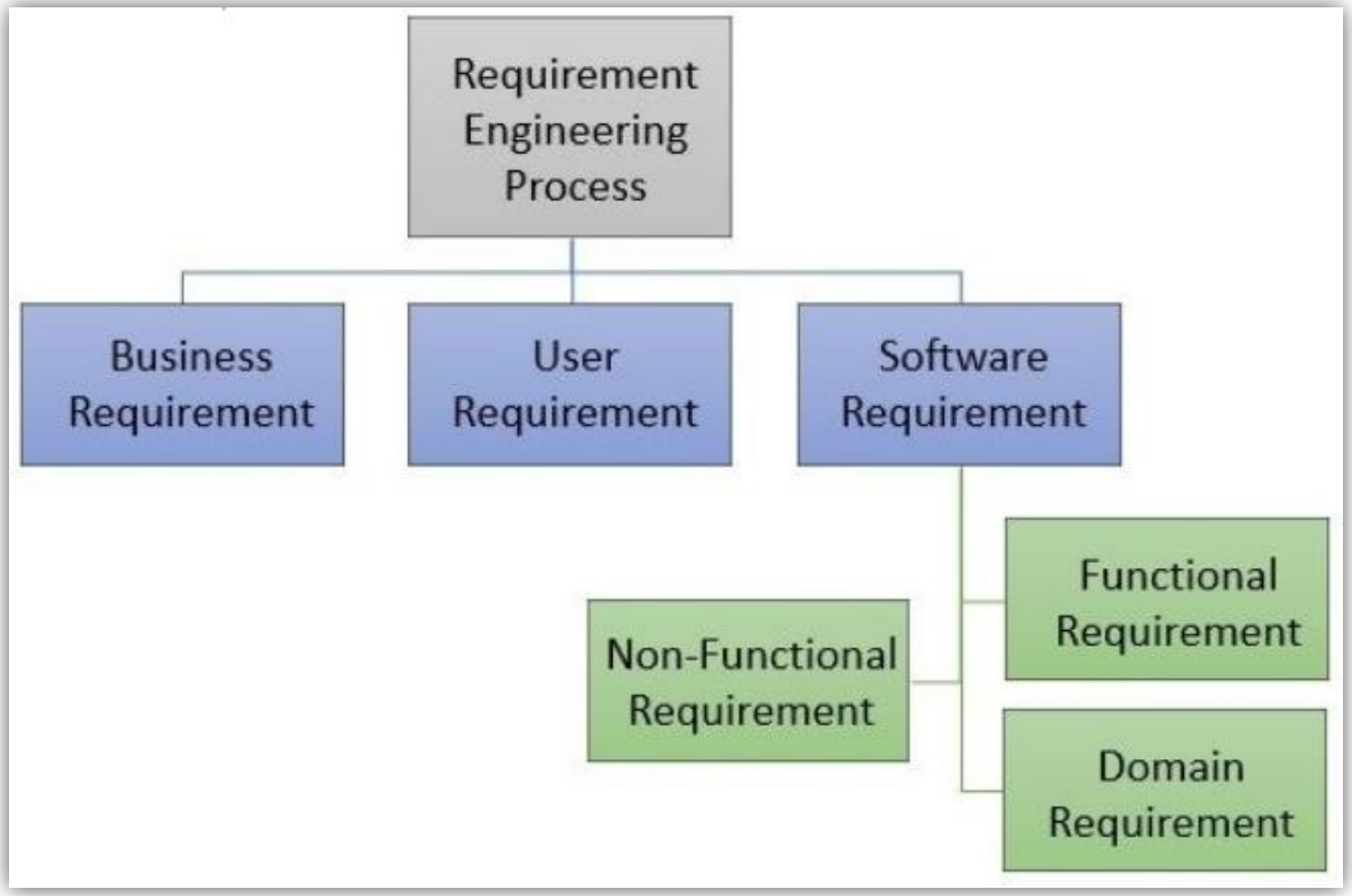
❑There is no universally accepted standard for user requirements statements, but here's one common format:

"The [user type] shall [interact with the software] in order to [meet a business goal or achieve a result]."

Software Requirements and its Types

❑**Software Requirements:** There are three types of Software requirements as follows:

1. Functional requirements
2. Non-Functional requirements
3. Domain requirements



Software Requirements and its Types

❑**Functional requirements** are such software requirements that are demanded explicitly as basic facilities of the system by the end-users.

❑So, these requirements for functionalities should be necessarily incorporated into the system as a part of the contract.

❑They describe system behavior under specific conditions, they are the functions that one can see directly in the final product, and it was the requirements of the users as well.

❑It describes a software system or its components.

❑These are represented as inputs to the software system, its behavior, and its output.

❑It can be a calculation, data manipulation, business process, user interaction, or any other specific functionality which defines what function a system is likely to perform.

❑A functional requirement can range from the high-level abstract statement of the sender's necessity to detailed mathematical functional requirement specifications.

1. Natural language

2. A structured or formatted language with no rigorous syntax and formal specification language with proper syntax.

Software Requirements and its Types

❑**Non-functional Requirements(NFRs)** requirements are defined as the quality constraints that the system must satisfy to complete the project contract.

❑But, the extent may vary to which implementation of these factors is done or get relaxed according to one project to another.

❑They are also called non-behavioral requirements or quality requirements/attributes.

❑They deal with issues like, Performance, Reusability, Flexibility, Reliability, Maintainability, Security, Portability

❑**Non-Functional Requirements are classified into many types.**

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

Software Requirements and its Types

❑**Domain requirements** are the requirements related to a particular category like software, purpose or industry, or other domain of projects.

❑Domain requirements can be functional or non-functional.

❑These are essential functions that a system of specific domains must necessarily exhibit.

❑The common factor for domain requirements is that they meet established standards or widely accepted feature sets for that category of the software project.

❑Domain requirements typically arise in military, medical, and financial industry sectors.

❑They are identified from that specific domain and are not user-specific.

1. **Software in medical equipment:** In medical equipment, software must be developed per IEC 60601 regarding medical electrical equipment's basic safety and performance. The software can be functional and usable but not acceptable for production because it fails to meet domain requirements.
2. **An Academic Software:** Such software must be developed to maintain records of an institute efficiently. Domain requirement of such software is the functionality of being able to access the list of faculty and list of students of each grade.

Requirement Engineering Process

❑**Requirements engineering (RE)** refers to the process of defining, documenting, and maintaining requirements in the engineering design process.

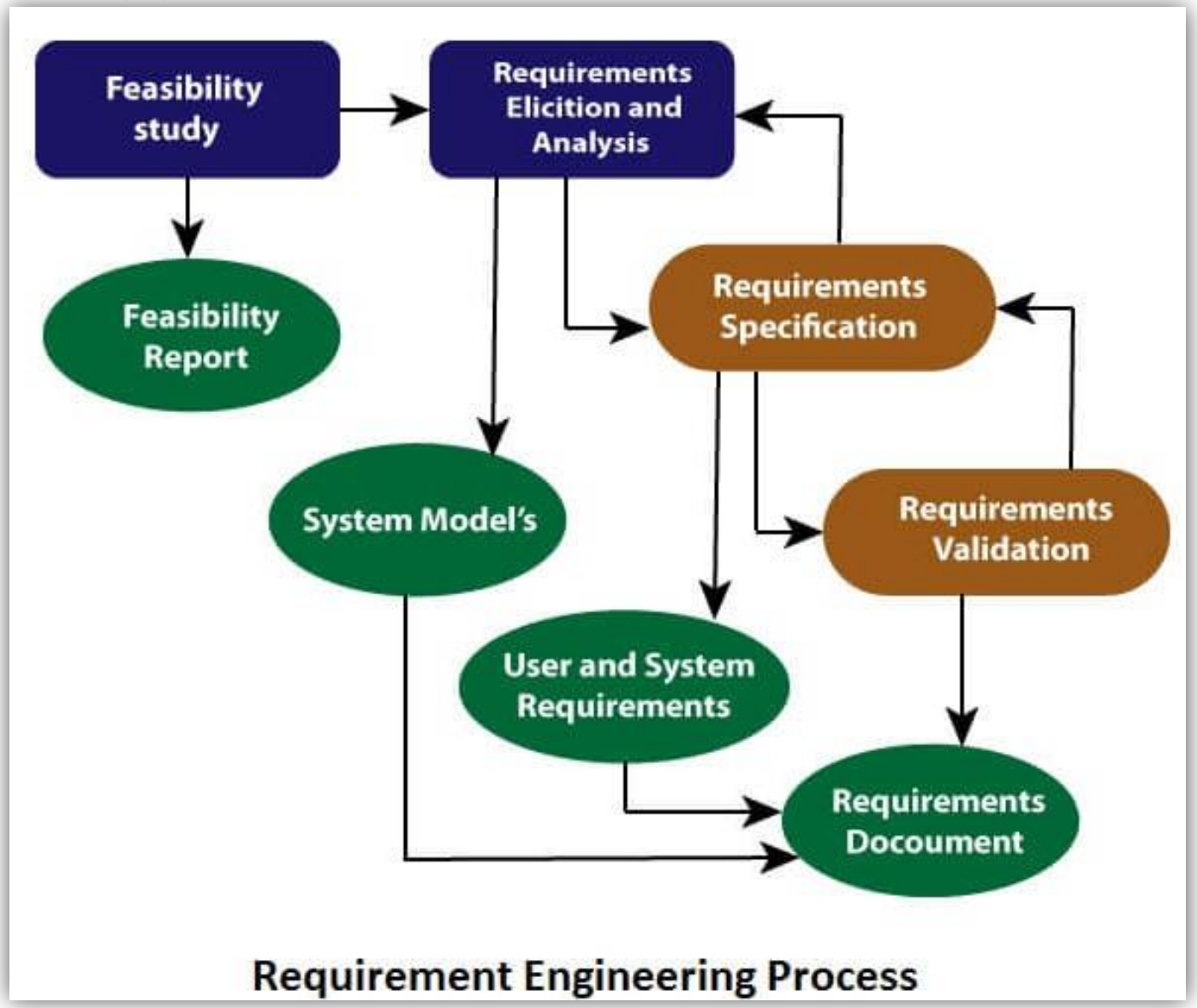
❑Requirement engineering provides the appropriate mechanism to understand what the customer desires, analyzing the need, and assessing feasibility, negotiating a reasonable solution, specifying the solution clearly, validating the specifications and managing the requirements as they are transformed into a working system.

❑Thus, requirement engineering is the disciplined application of proven principles, methods, tools, and notation to describe a proposed system's intended behavior and its associated constraints.

Requirement Engineering Process

❑**Requirements engineering (RE)** is a four-step process, which includes:

1. Feasibility Study
2. Requirement Elicitation and Analysis
3. Software Requirement Specification
4. Software Requirement Validation
5. Software Requirement Management



Requirement Engineering Process

Requirement Engineering Process

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

I. **Technical Feasibility:** evaluates the current technologies, which are needed to accomplish customer requirements within the time and budget.

II. **Operational Feasibility:** assesses the range in which the required software performs a series of levels to solve business problems and customer requirements.

III. **Economic Feasibility:** decides whether the necessary software can generate financial profits for an organization.

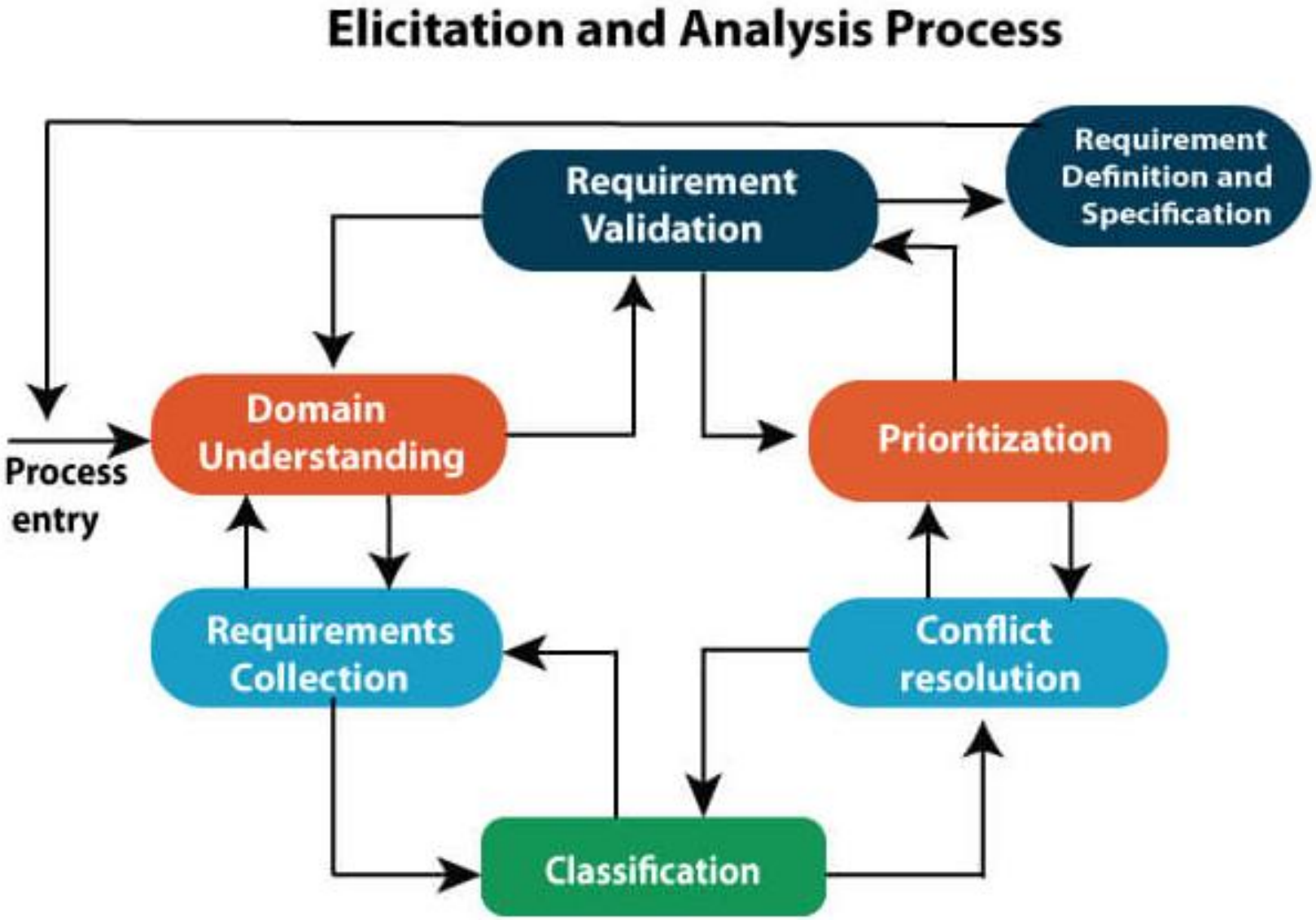
Requirement Engineering Process

2. Requirement Elicitation and Analysis: This is also known as the **gathering of requirements**.

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

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

Requirement Engineering Process



3. Software Requirement Specification:

- Software requirement specification is a kind of document which is created by a software analyst after the requirements collected from the various sources - the requirement received by the customer written in ordinary language.
- It is the job of the analyst to write the requirement in technical language so that they can be understood and beneficial by the development team.
- The models used at this stage include ER diagrams, data flow diagrams (DFDs), function decomposition diagrams (FDDs), data dictionaries, etc.

Requirement Engineering Process

3. Software Requirement Specification:

I. **Data Flow Diagrams:** Data Flow Diagrams (DFDs) are used widely for modeling the requirements. DFD shows the flow of data through a system. The system may be a company, an organization, a set of procedures, a computer hardware system, a software system, or any combination of the preceding. The DFD is also known as a data flow graph or bubble chart.

II. **Data Dictionaries:** Data Dictionaries are simply repositories to store information about all data items defined in DFDs. At the requirements stage, the data dictionary should at least define customer data items, to ensure that the customer and developers use the same definition and terminologies.

III. **Entity-Relationship Diagrams:** Another tool for requirement specification is the entity-relationship diagram, often called an "***E-R diagram***." It is a detailed logical representation of the data for the organization and uses three main constructs i.e. data entities, relationships, and their associated attributes.

Requirement Engineering Process

4. Software Requirement Validation:

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

4. Software Requirement Validation:

- Following are Requirements Validation Techniques

I. **Requirements reviews/inspections:** systematic manual analysis of the requirements.

II. **Prototyping:** Using an executable model of the system to check requirements.

III. **Test-case generation:** Developing tests for requirements to check testability.

IV. **Automated consistency analysis:** checking for the consistency of structured requirements descriptions.

Requirement Engineering Process

5. Software Requirement Management:

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

Requirement Elicitation

- **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.

Requirement Elicitation

❑**Requirement Elicitation:** Requirements elicitation is perhaps the most difficult, most error-prone, and most communication-intensive software development.

1. It can be successful only through an effective customer-developer partnership. It is needed to know what the users require.
2. Requirements elicitation involves the identification, collection, analysis, and refinement of the requirements for a software system.
3. It is a critical part of the software development life cycle and is typically performed at the beginning of the project.
4. Requirements elicitation involves stakeholders from different areas of the organization, including business owners, end-users, and technical experts.
5. The output of the requirements elicitation process is a set of clear, concise, and well-defined requirements that serve as the basis for the design and development of the software system.

Requirement Elicitation

❑Importance of Requirements Elicitation:

1. **Compliance with Business Objectives:** The process of elicitation guarantees that the software development endeavours 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.
2. **User Satisfaction:** It is easier to create software that fulfils 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.
3. **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.

Requirement Elicitation

❑Importance of Requirements Elicitation:

4. **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.
5. **Traceability and Documentation:** Throughout the software development process, traceability is based on requirements that are well-documented. Traceability helps with testing, validation, and maintenance by ensuring that every part of the software can be linked to a particular requirement.

Requirement Elicitation

❑**Requirements Elicitation Activities:** Requirements elicitation includes the subsequent activities. A few of them are listed below:

1. Knowledge of the overall area where the systems are applied.
2. The details of the precise customer problem where the system is going to be applied must be understood.
3. Interaction of system with external requirements.
4. Detailed investigation of user needs.
5. Define the constraints for system development.

Requirement Elicitation

❑**Requirements Elicitation Methods:** There are a number of requirements elicitation methods. Few of them are listed below:

1. **Interviews**
2. **Brainstorming Sessions**
3. **Facilitated Application Specification Technique**
4. **Quality Function Deployment**
5. **Use Case Approach**

Requirement Elicitation

❑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 maybe be open-ended or structured.

I. In open-ended interviews there is no pre-set agenda. Context free questions may be asked to understand the problem.

II. In a structured interview, an agenda of fairly open questions is prepared. Sometimes a proper questionnaire is designed for the interview.

Requirement Elicitation

❑Brainstorming Sessions

I. It is a group technique

II. It is intended to generate lots of new ideas hence providing a platform to share views

III. A highly trained facilitator is required to handle group bias and group conflicts.

IV. Every idea is documented so that everyone can see it.

V. Finally, a document is prepared which consists of the list of requirements and their priority if possible.

Requirement Elicitation

❑Facilitated Application Specification Technique

❑Its objective is to bridge the expectation gap – the difference between what the developers think they are supposed to build and what customers think they are going to get.

❑A team-oriented approach is developed for requirements gathering.

❑Each attendee is asked to make a list of objects that are:

I. Part of the environment that surrounds the system.

II. Produced by the system.

III. Used by the system.

❑Each participant prepares his/her list, different lists are then combined, redundant entries are eliminated, team is divided into smaller sub-teams to develop mini-specifications and finally a draft of specifications is written down using all the inputs from the meeting.

Requirement Elicitation

❑**Quality Function Deployment:** In this technique customer satisfaction is of prime concern, hence it emphasizes on the requirements which are valuable to the customer. 3 types of requirements are identified:

I. **Normal requirements:** In this the objective and goals of the proposed software are discussed with the customer. Example – normal requirements for a result management system may be entry of marks, calculation of results, etc.

II. **Expected requirements:** These requirements are so obvious that the customer need not explicitly state them. Example – protection from unauthorized access.

III. **Exciting requirements:** It includes features that are beyond customer's expectations and prove to be very satisfying when present. Example – when unauthorized access is detected, it should backup and shutdown all processes.

Requirement Elicitation

❑**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 include three major things – Actor, use cases, use case diagram.

I. **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.

II. **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.

III. **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.
   - A stick figure is used to represent an actor.
   - An oval is used to represent a use case.
   - A line is used to represent a relationship between an actor and a use case.

Requirement Elicitation

❑Steps Of Requirements Elicitation:

I. Identify all the stakeholders, e.g., Users, developers, customers etc.

II. List out all requirements from customer.

III. A value indicating degree of importance is assigned to each requirement.

IV. In the end the final list of requirements is categorized as:

- It is possible to achieve.
- It should be deferred and the reason for it.
- It is impossible to achieve and should be dropped off.

Requirement Elicitation

❑Advantages of Requirements Elicitation:

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

Requirement Elicitation

❑Disadvantages of Requirements Elicitation:

1. **Time consuming:** Can be time-consuming and expensive.
2. **Skills required:** Requires specialized skills and expertise.
3. **Impacted by changing requirements:** May be impacted by changing business needs and requirements.
4. **Impacted by other factors:** Can be impacted by political and organizational factors.
5. **Lack of commitment from stakeholders:** Can result in a lack of buy-in and commitment from stakeholders.
6. **Impacted by conflicting priorities:** Can be impacted by conflicting priorities and competing interests.
7. **Sometimes inaccurate requirements:** May result in incomplete or inaccurate requirements if not properly managed.

Software Requirement Specification

- The production of the requirements stage of the software development process is **Software Requirements Specifications (SRS)** (also called a **requirements document**).
- This report lays a foundation for software engineering activities and is constructing when entire requirements are elicited and analyzed.
- **SRS** is a formal report, which acts as a representation of software that enables the customers to review whether it (SRS) is according to their requirements.
- Also, it comprises user requirements for a system as well as detailed specifications of the system requirements.

Software Requirement Specification

- The SRS is a specification for a specific software product, program, or set of applications that perform particular functions in a specific environment.
- It serves several goals depending on who is writing it.
- First, the SRS could be written by the client of a system.
- Second, the SRS could be written by a developer of the system.
- The two methods create entirely various situations and establish different purposes for the document altogether.
- The first case, SRS, is used to define the needs and expectation of the users.
- The second case, SRS, is written for various purposes and serves as a contract document between customer and developer.

Software Requirement Specification

- Characteristics of good SRS

Software Requirement Specification

❑**Following are the features of a good SRS document:**

1. Correctness: User review is used to provide the accuracy of requirements stated in the SRS. SRS is said to be perfect if it covers all the needs that are truly expected from the system.

2. Completeness: The SRS is complete if, and only if, it includes the following elements:

(1). All essential requirements, whether relating to functionality, performance, design, constraints, attributes, or external interfaces.

(2). Definition of their responses of the software to all realizable classes of input data in all available categories of situations.

(3). Full labels and references to all figures, tables, and diagrams in the SRS and definitions of all terms and units of measure.

Software Requirement Specification

❑**Following are the features of a good SRS document:**

3. Consistency: The SRS is consistent if, and only if, no subset of individual requirements described in its conflict. There are three types of possible conflict in the SRS:

(1). The specified characteristics of real-world objects may conflicts. For example,

a) The format of an output report may be described in one requirement as tabular but in another as textual.

(b) One condition may state that all lights shall be green while another states that all lights shall be blue.

(2). There may be a reasonable or temporal conflict between the two specified actions. For example,

(a) One requirement may determine that the program will add two inputs, and another may determine that the program will multiply them.

(b) One condition may state that "A" must always follow "B," while other requires that "A and B" co-occurs.

(3). Two or more requirements may define the same real-world object but use different terms for that object. For example, a program's request for user input may be called a "prompt" in one requirement's and a "cue" in another. The use of standard terminology and descriptions promotes consistency.

❑**Following are the features of a good SRS document:**

4. Unambiguousness: SRS is unambiguous when every fixed requirement has only one interpretation. This suggests that each element is uniquely interpreted. In case there is a method used with multiple definitions, the requirements report should determine the implications in the SRS so that it is clear and simple to understand.

5. Ranking for importance and stability: The SRS is ranked for importance and stability if each requirement in it has an identifier to indicate either the significance or stability of that particular requirement.

6. Modifiability: SRS should be made as modifiable as likely and should be capable of quickly obtain changes to the system to some extent. Modifications should be perfectly indexed and cross-referenced.

Software Requirement Specification

❑**Following are the features of a good SRS document:**

7. Verifiability: SRS is correct when the specified requirements can be verified with a cost-effective system to check whether the final software meets those requirements. The requirements are verified with the help of reviews.

8. Traceability: The SRS is traceable if the origin of each of the requirements is clear and if it facilitates the referencing of each condition in future development or enhancement documentation

9. Design Independence: There should be an option to select from multiple design alternatives for the final system. More specifically, the SRS should not contain any implementation details.

10. Testability: An SRS should be written in such a method that it is simple to generate test cases and test plans from the report.

❑**Following are the features of a good SRS document:**

11. Understandable by the customer: An end user may be an expert in his/her explicit domain but might not be trained in computer science. Hence, the purpose of formal notations and symbols should be avoided too as much extent as possible. The language should be kept simple and clear.

12. The right level of abstraction: If the SRS is written for the requirements stage, the details should be explained explicitly. Whereas, for a feasibility study, fewer analysis can be used. Hence, the level of abstraction modifies according to the objective of the SRS.

Software Requirement Specification

❑**Following are the Properties of a good SRS document:**

- **Concise:** The SRS report should be concise and at the same time, unambiguous, consistent, and complete. Verbose and irrelevant descriptions decrease readability and also increase error possibilities.
- **Structured:** It should be well-structured. A well-structured document is simple to understand and modify. In practice, the SRS document undergoes several revisions to cope up with the user requirements. Often, user requirements evolve over a period of time. Therefore, to make the modifications to the SRS document easy, it is vital to make the report well-structured.
- **Black-box view:** It should only define what the system should do and refrain from stating how to do these. This means that the SRS document should define the external behavior of the system and not discuss the implementation issues. The SRS report should view the system to be developed as a black box and should define the externally visible behavior of the system. For this reason, the SRS report is also known as the black-box specification of a system.

Software Requirement Specification

❑**Following are the Properties of a good SRS document:**

- **Conceptual integrity:** It should show conceptual integrity so that the reader can merely understand it. Response to undesired events: It should characterize acceptable responses to unwanted events. These are called system response to exceptional conditions.
- **Verifiable:** All requirements of the system, as documented in the SRS document, should be correct. This means that it should be possible to decide whether or not requirements have been met in an implementation.

System Modeling: Requirement Specification and Requirement Validation

- System modeling is the process of developing abstract models of a system, with each model presenting a different view or perspective of that system.
- It is about representing a system using some kind of graphical notation, which is now almost always based on notations in the Unified Modeling Language (UML).
- Models help the analyst to understand the functionality of the system; they are used to communicate with customers.

System Modeling: Requirement Specification and Requirement Validation

❑Models can explain the system from **different perspectives**:

1. An **external** perspective, where you model the context or environment of the system.
2. An **interaction** perspective, where you model the interactions between a system and its environment, or between the components of a system.
3. A **structural** perspective, where you model the organization of a system or the structure of the data that is processed by the system.
4. A **behavioral** perspective, where you model the dynamic behavior of the system and how it responds to events.

System Modeling: Requirement Specification and Requirement Validation

❑Five types of UML diagrams that are the most useful for system modeling:

1. **Activity** diagrams, which show the activities involved in a process or in data processing.
2. **Use case** diagrams, which show the interactions between a system and its environment.
3. **Sequence** diagrams, which show interactions between actors and the system and between system components.
4. **Class** diagrams, which show the object classes in the system and the associations between these classes.
5. **State** diagrams, which show how the system reacts to internal and external events.

System Modeling: Requirement Specification and Requirement Validation

- Models of both new and existing system are used during requirements engineering.
- Models of the existing systems help clarify what the existing system does and can be used as a basis for discussing its strengths and weaknesses.
- These then lead to requirements for the new system.
- Models of the new system are used during requirements engineering to help explain the proposed requirements to other system stakeholders.
- Engineers use these models to discuss design proposals and to document the system for implementation.

CASE TOOLs – UML 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.

CASE TOOLs – UML Class Diagram

- The **main purpose of class diagrams** is to build a static view of an application.
- It is the only diagram that is widely used for construction, and it can be mapped with object-oriented languages.
- It is one of the most popular UML diagrams.
- Following are the purpose of class diagrams given below:

1. It analyses and designs a static view of an application.
2. It describes the major responsibilities of a system.
3. It is a base for component and deployment diagrams.
4. It incorporates forward and reverse engineering.

CASE TOOLs – UML Class Diagram

❑Benefits of Class Diagrams

1. It can represent the object model for complex systems.
2. It reduces the maintenance time by providing an overview of how an application is structured before coding.
3. It provides a general schematic of an application for better understanding.
4. It represents a detailed chart by highlighting the desired code, which is to be programmed.
5. It is helpful for the stakeholders and the developers.

CASE TOOLs – UML Class Diagram

- **Vital components of a Class Diagram**: The class diagram is made up of three sections:
- **Upper Section**
- **Middle Section**
- **Lower Section**

CASE TOOLs – UML Class Diagram

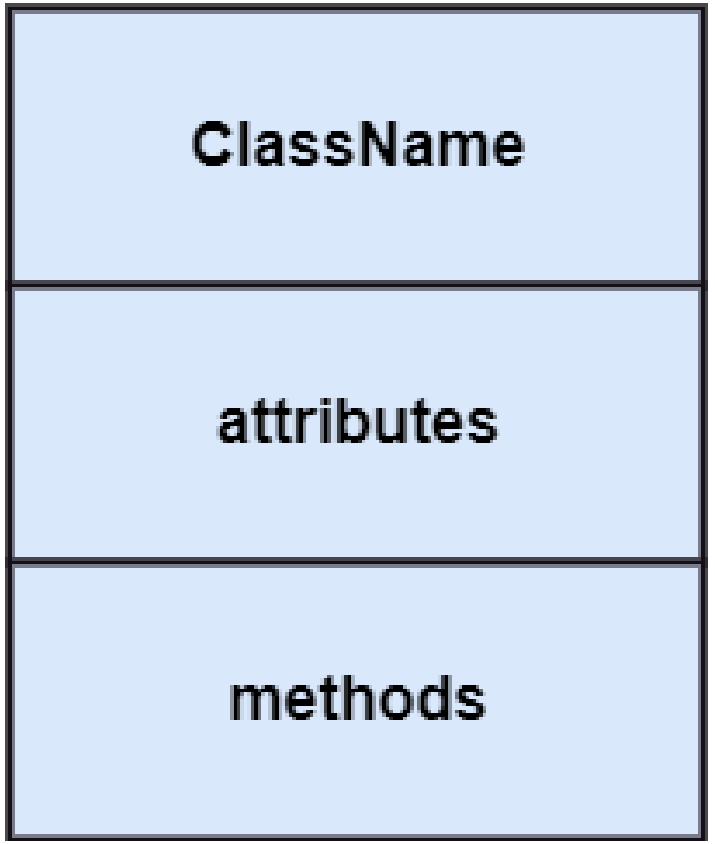
- **Upper Section:** The upper section encompasses the name of the class.
- A class is a representation of similar objects that shares the same relationships, attributes, operations, and semantics.
- Some of the following rules that should be taken into account while representing a class are given below:
  - Capitalize the initial letter of the class name.
  - Place the class name in the center of the upper section.
  - A class name must be written in bold format.
  - The name of the abstract class should be written in italics format.

CASE TOOLs – UML Class Diagram

- **Middle Section:** The middle section constitutes the attributes, which describe the quality of the class.
- The attributes have the following characteristics:
  - The attributes are written along with its visibility factors, which are public (+), private (-), protected (#), and package (~).
  - The accessibility of an attribute class is illustrated by the visibility factors.
  - A meaningful name should be assigned to the attribute, which will explain its usage inside the class.

CASE TOOLs – UML Class Diagram

- **Lower Section:** The lower section contain methods or operations.
- The methods are represented in the form of a list, where each method is written in a single line.
- It demonstrates how a class interacts with data.

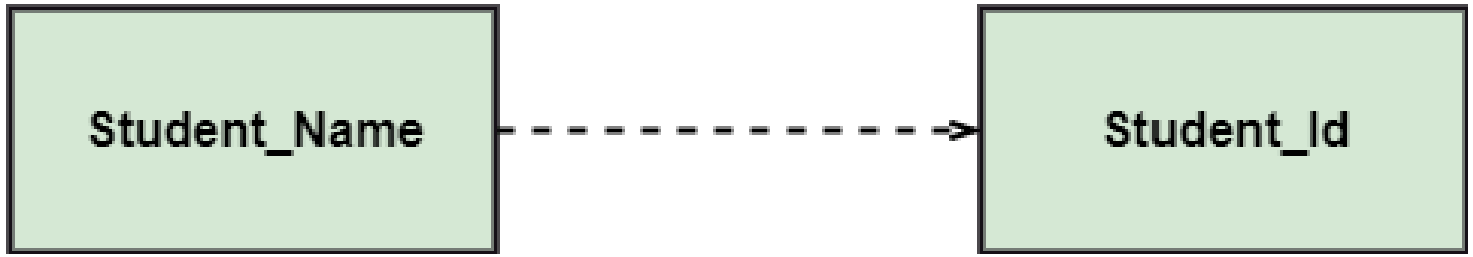


CASE TOOLs – UML Class Diagram

- **Relationships:** Following are the relationship in UML Class Diagram.
- Dependency:
- Generalization:
- Association:
- Multiplicity:
- Aggregation:
- Composition:

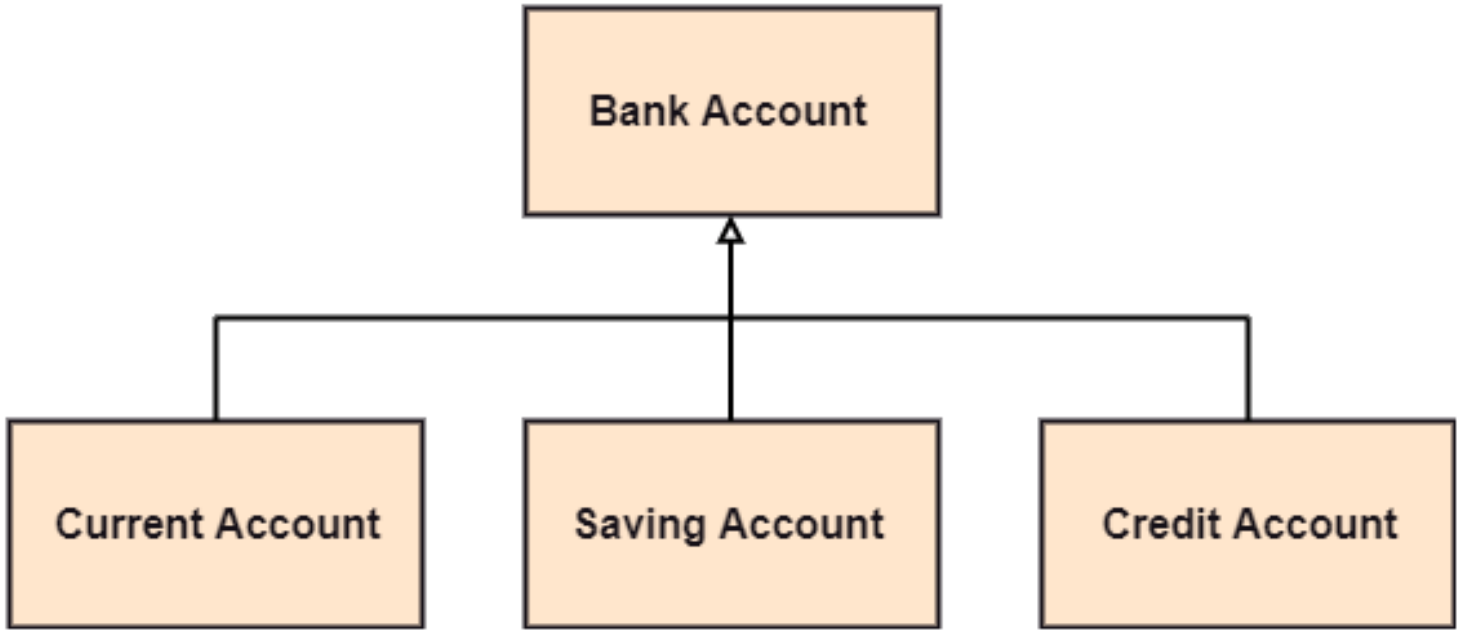
❑Relationships:

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



❑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. For example, The Current Account, Saving Account, and Credit Account are the generalized form of Bank Account.

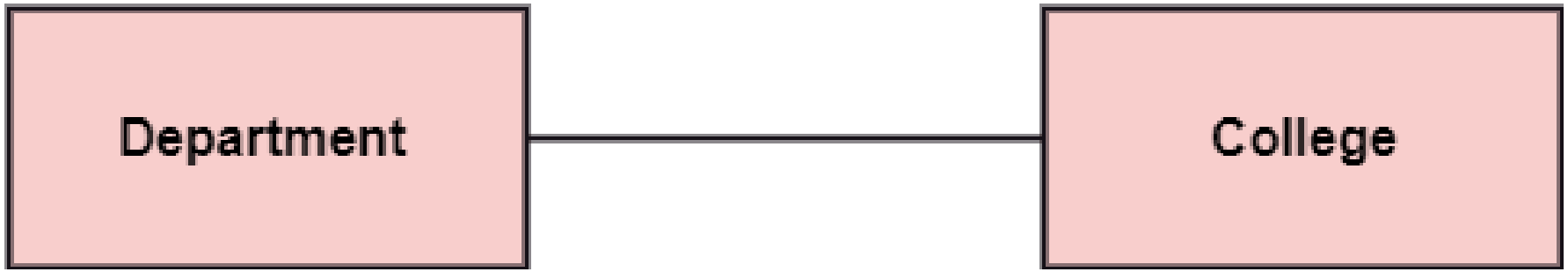


CASE TOOLs – UML Class Diagram

❑**Relationships:**

❑**Association:** It describes a static or physical connection between two or more objects. It depicts how many objects are there in the relationship.

❑For example, a department is associated with the college.

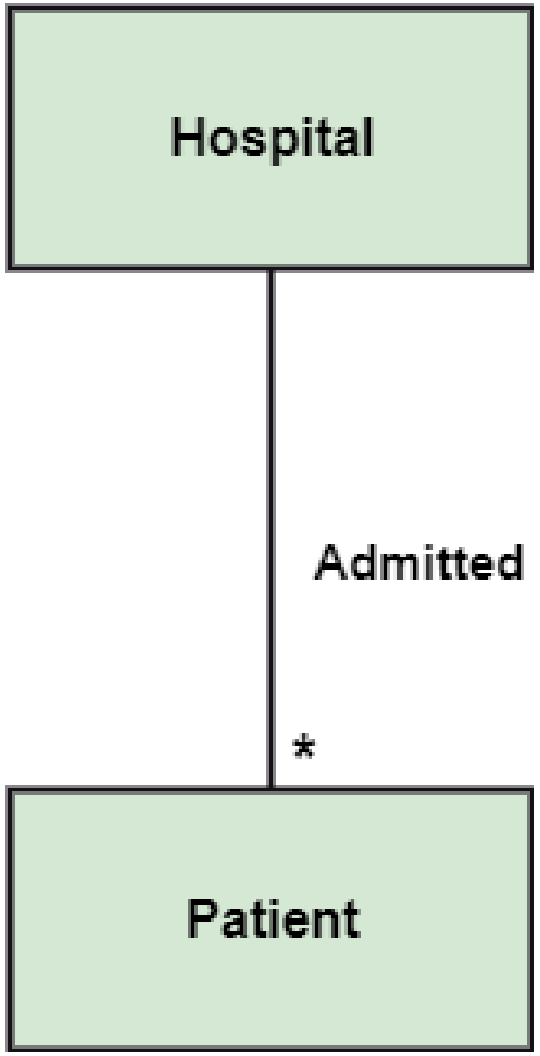


CASE TOOLs – UML Class Diagram

❑**Relationships:**

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

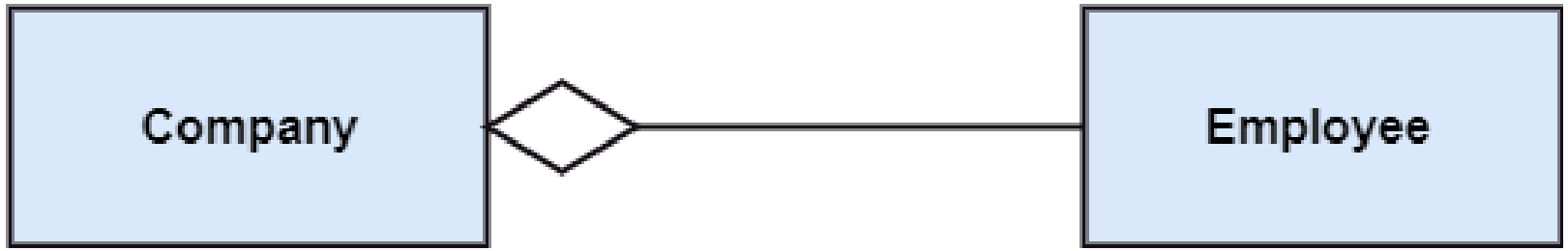
❑For example, multiple patients are admitted to one hospital.



❑Relationships:

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

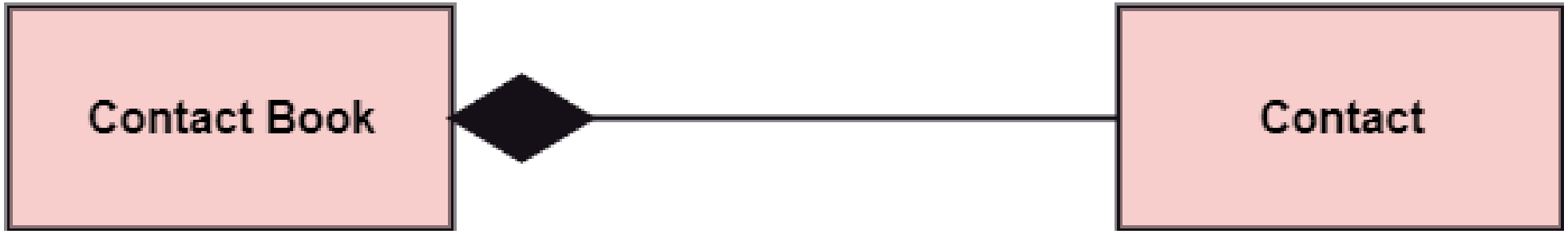
❑The company encompasses a number of employees, and even if one employee resigns, the company still exists.



❑Relationships:

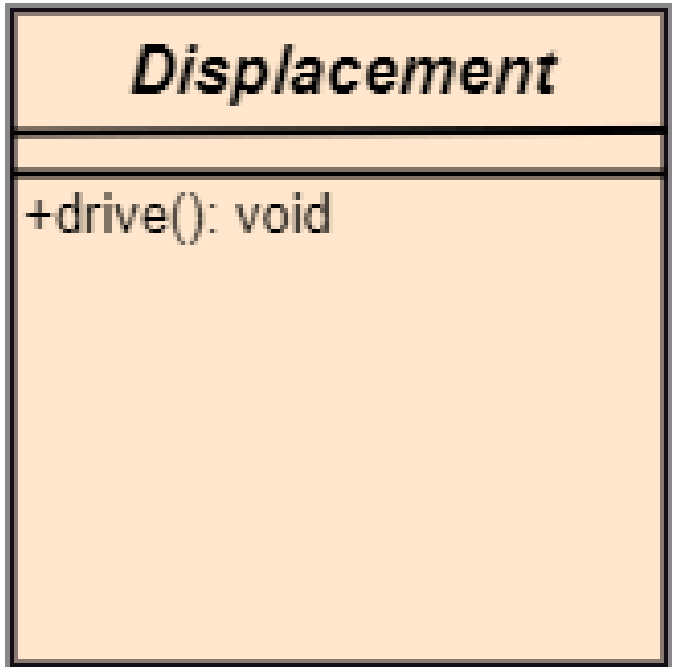
❑**Composition:** The composition is a subset of aggregation. It portrays the dependency between the parent and its child, which means if one part is deleted, then the other part also gets discarded. It represents a whole-part relationship.

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



CASE TOOLs – UML Class Diagram

❑**Abstract Classes:** In the abstract class, no objects can be a direct entity of the abstract class. The abstract class can neither be declared nor be instantiated. It is used to find the functionalities across the classes. The notation of the abstract class is similar to that of class; the only difference is that the name of the class is written in italics. Since it does not involve any implementation for a given function, it is best to use the abstract class with multiple objects.



❑Let us assume that we have an abstract class named **displacement** with a method declared inside it, and that method will be called as a **drive ()**. Now, this abstract class method can be implemented by any object, for example, car, bike, scooter, cycle, etc.

CASE TOOLs – UML Class Diagram

❑How to draw a Class Diagram?

❑The class diagram is used most widely to construct software applications.

❑It not only represents a static view of the system but also all the major aspects of an application.

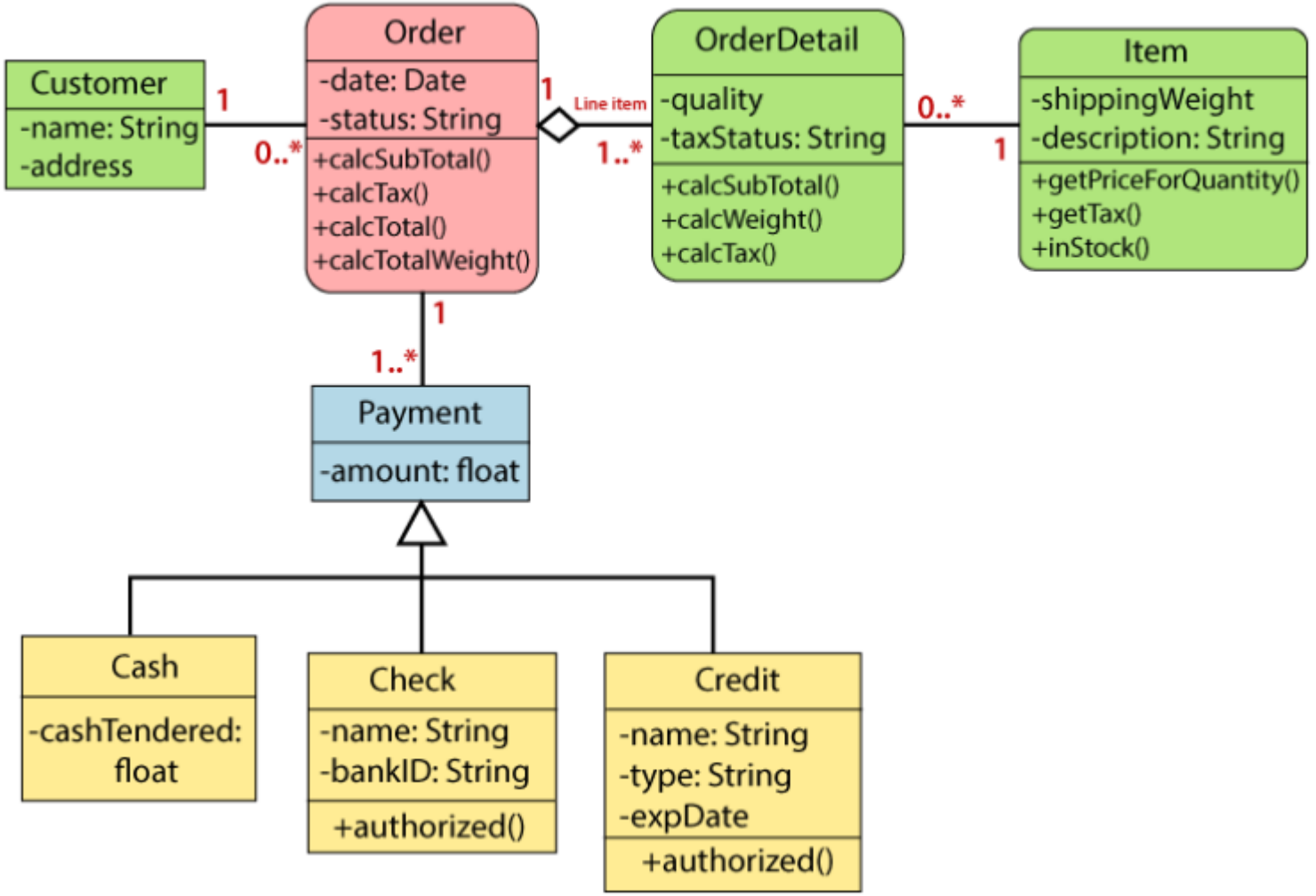
❑A collection of class diagrams as a whole represents a system.

❑Some key points that are needed to keep in mind while drawing a class diagram are given below:

1. **To describe a complete aspect of the system, it is suggested to give a meaningful name to the class diagram.**
2. **The objects and their relationships should be acknowledged in advance.**
3. **The attributes and methods (responsibilities) of each class must be known.**
4. **A minimum number of desired properties should be specified as more number of the unwanted property will lead to a complex diagram.**
5. **Notes can be used as and when required by the developer to describe the aspects of a diagram.**
6. **The diagrams should be redrawn and reworked as many times to make it correct before producing its final version.**

CASE TOOLs – UML Class Diagram

❑**Class Diagram Example:** A class diagram describing the sales order system is given below.



CASE TOOLs – UML Use Case Diagram

- A use case diagram is used to represent the dynamic behavior of a system.
- It encapsulates the system's functionality by incorporating use cases, actors, and their relationships.
- It models the tasks, services, and functions required by a system/subsystem of an application.
- It depicts the high-level functionality of a system and also tells how the user handles a system.

CASE TOOLs – UML Use Case Diagram

❑Purpose of Use Case Diagrams

- The main purpose of a use case diagram is to portray the dynamic aspect of a system.
- It accumulates the system's requirement, which includes both internal as well as external influences.
- It invokes persons, use cases, and several things that invoke the actors and elements accountable for the implementation of use case diagrams.
- It represents how an entity from the external environment can interact with a part of the system.
- Following are the purposes of a use case diagram given below:

1. It gathers the system's needs.
2. It depicts the external view of the system.
3. It recognizes the internal as well as external factors that influence the system.
4. It represents the interaction between the actors.

CASE TOOLs – UML Use Case Diagram

❑How to draw a Use Case diagram?

❑It is essential to analyze the whole system before starting with drawing a use case diagram, and then the system's functionalities are found.

❑And once every single functionality is identified, they are then transformed into the use cases to be used in the use case diagram.

❑After that, we will enlist the actors that will interact with the system.

❑The actors are the person or a thing that invokes the functionality of a system.

❑It may be a system or a private entity, such that it requires an entity to be pertinent to the functionalities of the system to which it is going to interact.

❑Once both the actors and use cases are enlisted, the relation between the actor and use case/ system is inspected.

❑It identifies the no of times an actor communicates with the system. Basically, an actor can interact multiple times with a use case or system at a particular instance of time.

CASE TOOLs – UML Use Case Diagram

❑How to draw a Use Case diagram?

❑Following are some rules that must be followed while drawing a use case diagram:

1. A pertinent and meaningful name should be assigned to the actor or a use case of a system.
2. The communication of an actor with a use case must be defined in an understandable way.
3. Specified notations to be used as and when required.
4. The most significant interactions should be represented among the multiple no of interactions between the use case and actors.

CASE TOOLs – UML Use Case Diagram

❑**Example of a Use Case Diagram:** A use case diagram depicting the Online Shopping website is given below.

❑Here the Web Customer actor makes use of any online shopping website to purchase online.

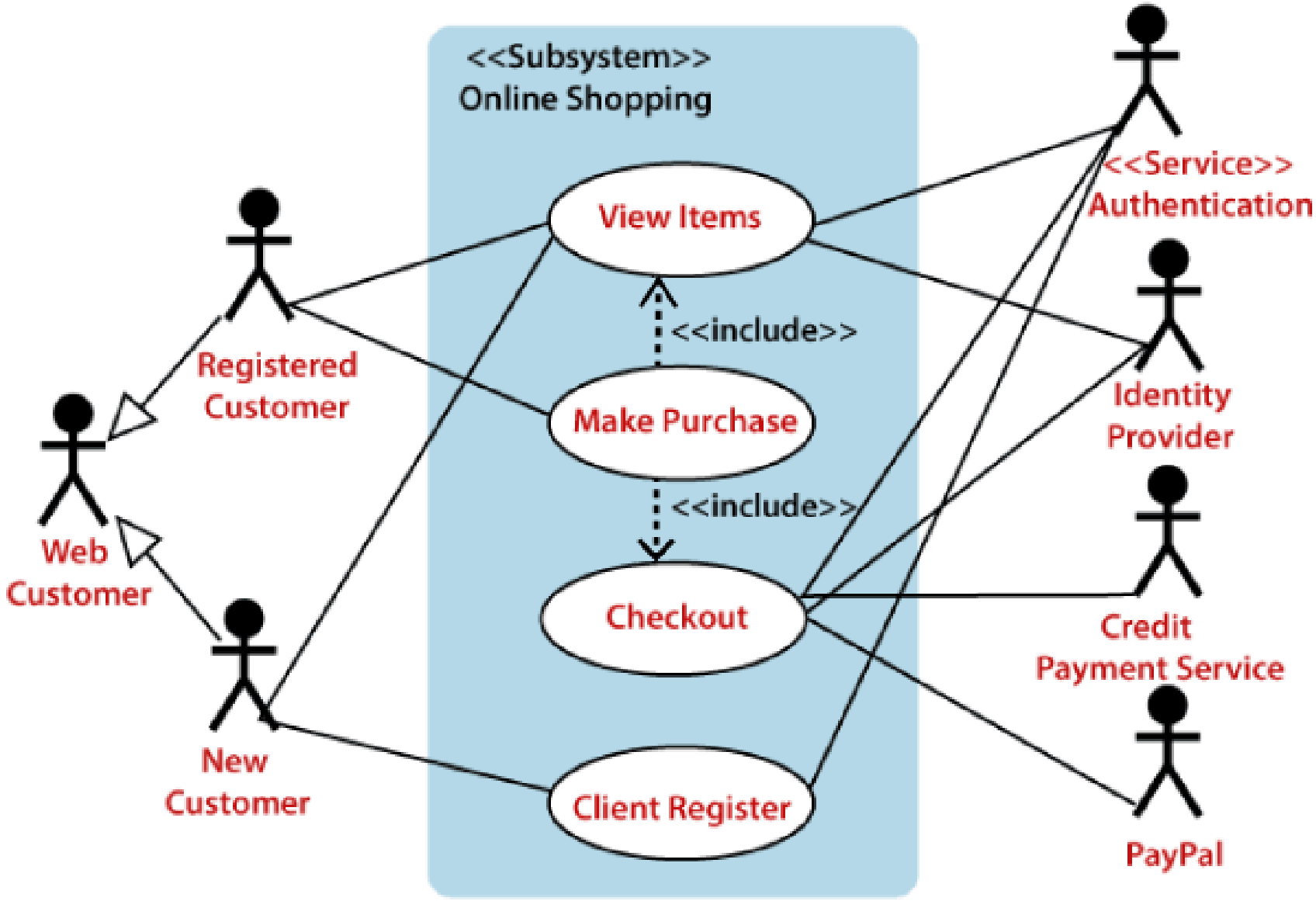
❑The top-level uses are as follows; View Items, Make Purchase, Checkout, Client Register.

❑The **View Items** use case is utilized by the customer who searches and view products.

❑The **Client Register** use case allows the customer to register itself with the website for availing gift vouchers, coupons, or getting a private sale invitation.

❑It is to be noted that the **Checkout** is an included use case, which is part of **Making Purchase,** and it is not available by itself.

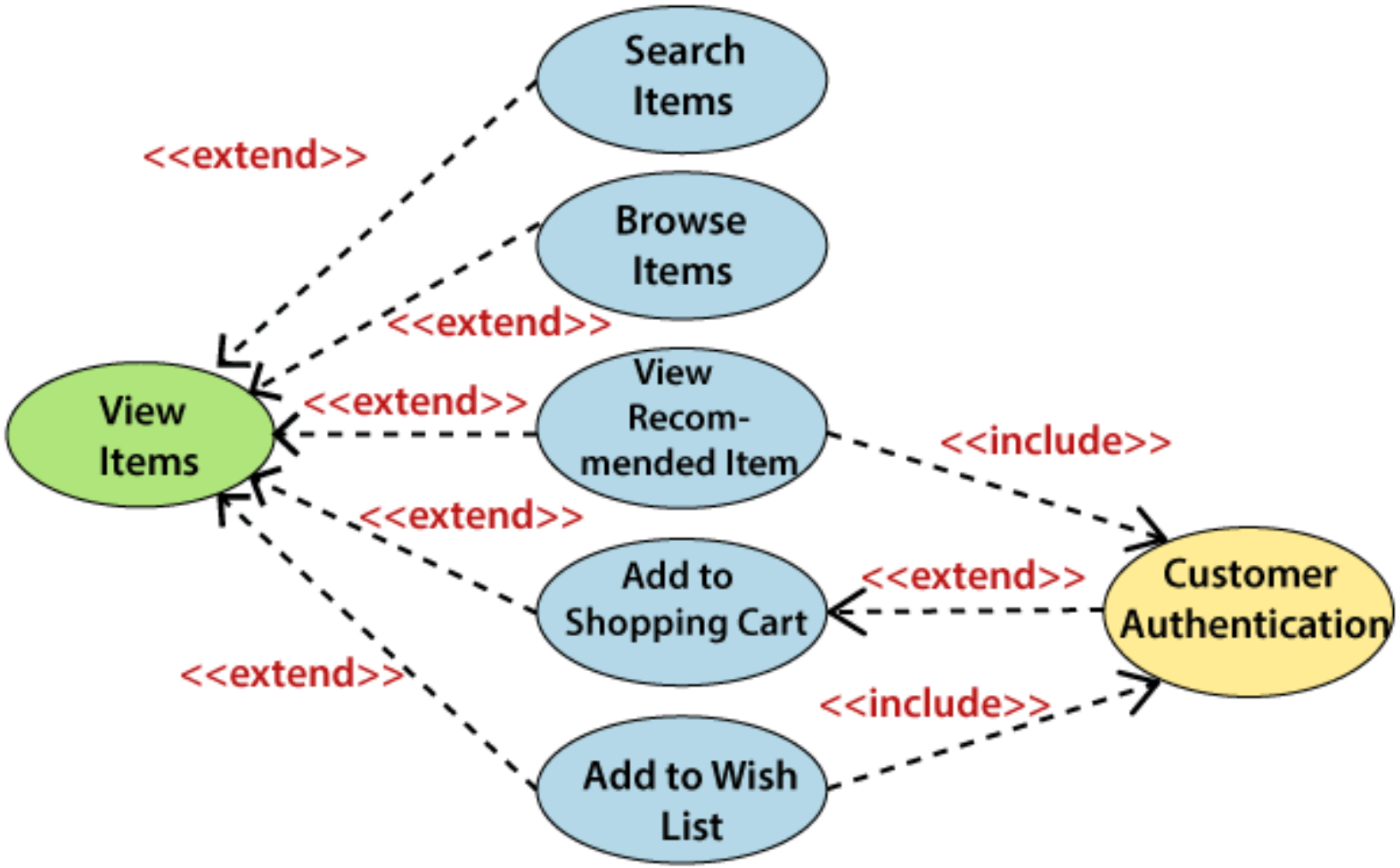
CASE TOOLs – UML Use Case Diagram



CASE TOOLs – UML Use Case Diagram

- The **View Items** is further extended by several use cases such as; Search Items, Browse Items, View Recommended Items, Add to Shopping Cart, Add to Wish list.
- All of these extended use cases provide some functions to customers, which allows them to search for an item.
- The View Items is further extended by several use cases such as; Search Items, Browse Items, View Recommended Items, Add to Shopping Cart, Add to Wish list.
- All of these extended use cases provide some functions to customers, which allows them to search for an item.
- Both **View Recommended Item** and **Add to Wish List** include the Customer Authentication use case, as they necessitate authenticated customers, and simultaneously item can be added to the shopping cart without any user authentication.

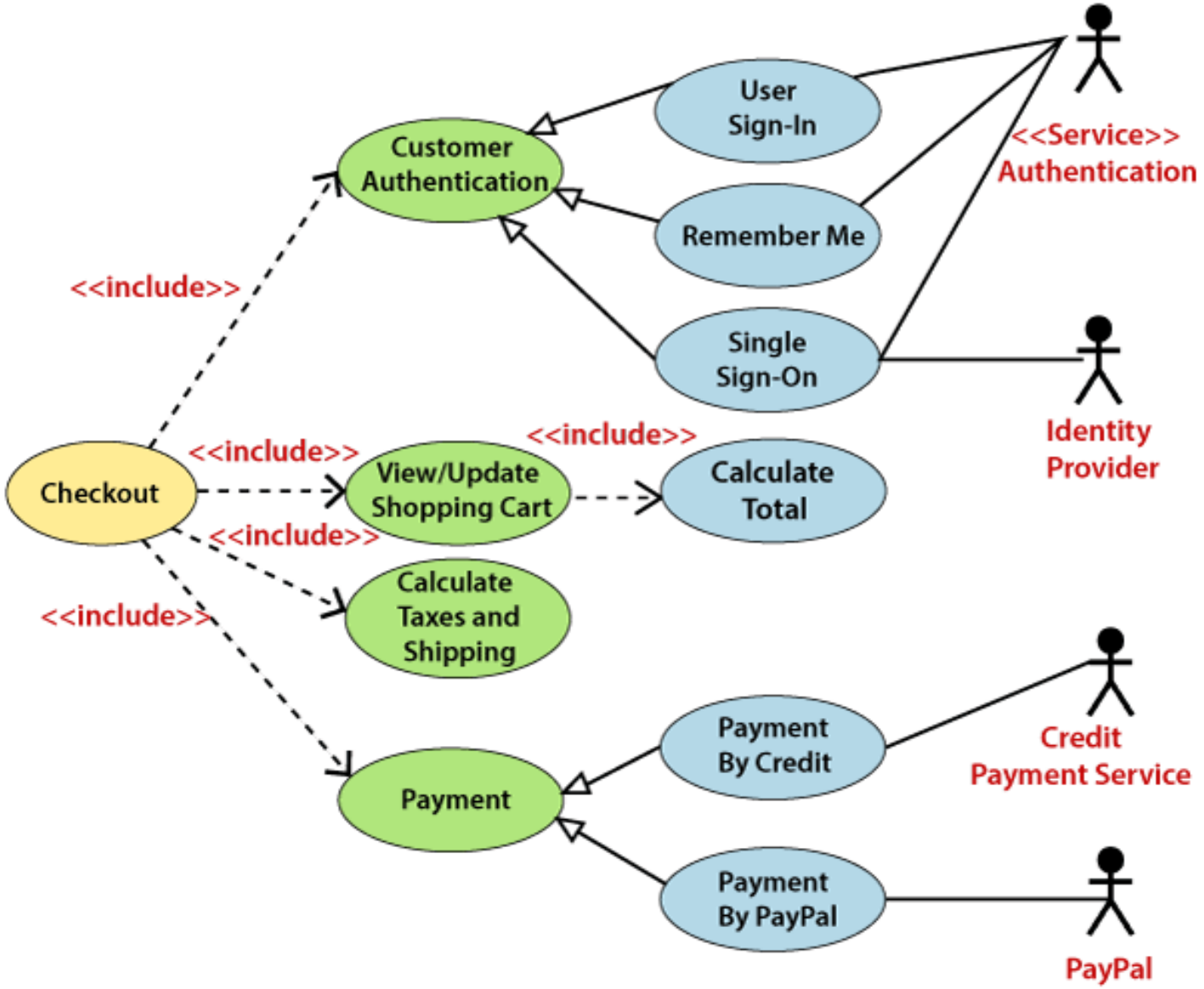
CASE TOOLs – UML Use Case Diagram



CASE TOOLs – UML Use Case Diagram

- Similarly, the **Checkout** use case also includes the following use cases, as shown below.
- It requires an authenticated Web Customer, which can be done by login page, user authentication cookie ("Remember me"), or Single Sign-On (SSO).
- SSO needs an external identity provider's participation, while Web site authentication service is utilized in all these use cases.
- The Checkout use case involves Payment use case that can be done either by the credit card and external credit payment services or with PayPal.

CASE TOOLs – UML Use Case Diagram



CASE TOOLs – UML Use Case Diagram

❑Tips for drawing a Use Case diagram: Following are some important tips that are to be kept in mind while drawing a use case diagram:

1. **A simple and complete use case diagram should be articulated.**
2. **A use case diagram should represent the most significant interaction among the multiple interactions.**
3. **At least one module of a system should be represented by the use case diagram.**
4. **If the use case diagram is large and more complex, then it should be drawn more generalized.**

CASE TOOLs – Sequence Diagram

- The sequence diagram represents the flow of messages in the system and is also termed as an event diagram.
- It helps in envisioning several dynamic scenarios.
- It portrays the communication between any two lifelines as a time-ordered sequence of events, such that these lifelines took part at the run time.
- In UML, the lifeline is represented by a vertical bar, whereas the message flow is represented by a vertical dotted line that extends across the bottom of the page.
- It incorporates the iterations as well as branching.

CASE TOOLs – Sequence Diagram

❑Purpose of a Sequence Diagram

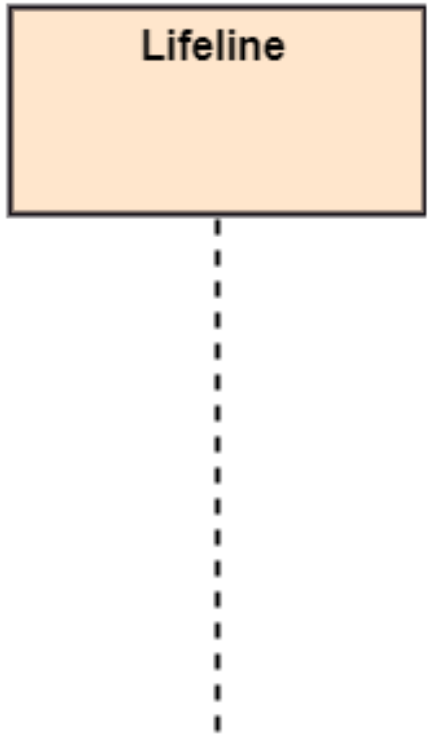
To model high-level interaction among active objects within a system.

To model interaction among objects inside a collaboration realizing a use case.

It either models generic interactions or some certain instances of interaction.

❑Notations of a Sequence Diagram:

❑**Lifeline:** An individual participant in the sequence diagram is represented by a lifeline. It is positioned at the top of the diagram.



CASE TOOLs – Sequence Diagram

❑Notations of a Sequence Diagram:

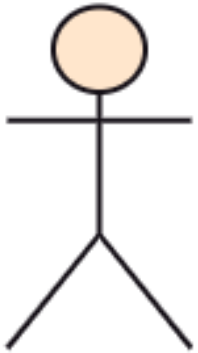
❑**Actor** : A role played by an entity that interacts with the subject is called as an actor.

❑ It is out of the scope of the system.

❑It represents the role, which involves human users and external hardware or subjects.

❑An actor may or may not represent a physical entity, but it purely depicts the role of an entity.

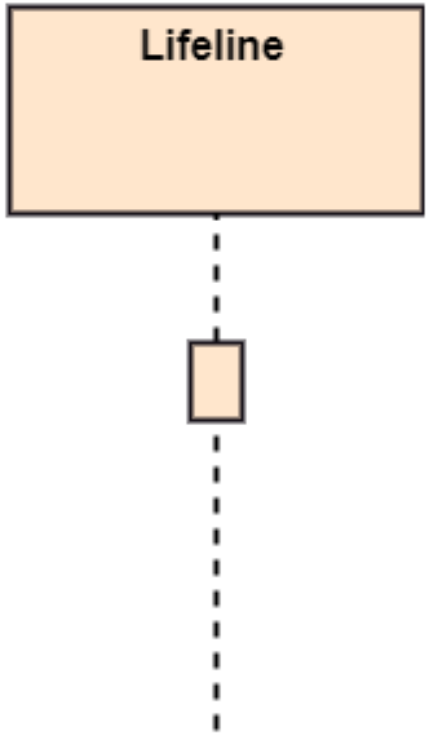
❑Several distinct roles can be played by an actor or vice versa.



Actor

❑Notations of a Sequence Diagram:

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



CASE TOOLs – Sequence Diagram

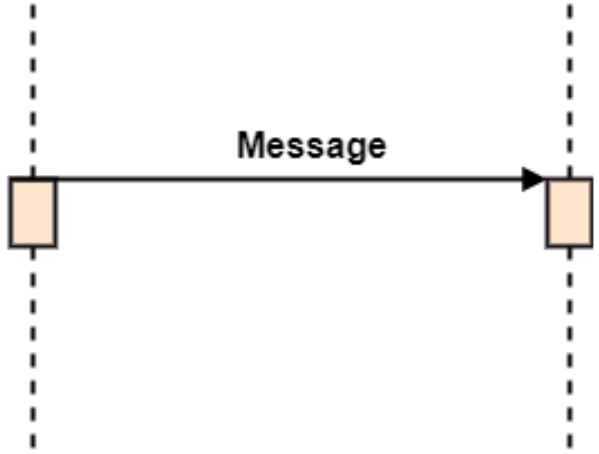
❑Notations of a Sequence Diagram:

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

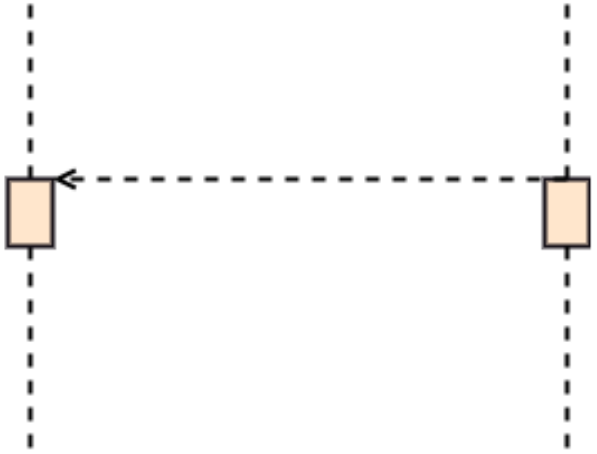
❑Notations of a Sequence Diagram:

❑Messages:

- **Call Message:** It defines a particular communication between the lifelines of an interaction, which represents that the target lifeline has invoked an operation.



- **Return Message:** It defines a particular communication between the lifelines of interaction that represent the flow of information from the receiver of the corresponding caller message.

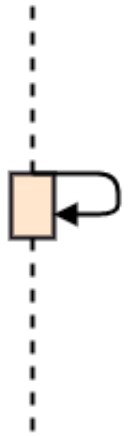


CASE TOOLs – Sequence Diagram

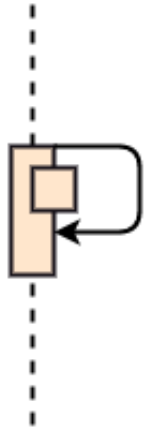
❑Notations of a Sequence Diagram:

❑Messages:

- **Self Message:** It describes a communication, particularly between the lifelines of an interaction that represents a message of the same lifeline, has been invoked.



- **Recursive Message:** A self message sent for recursive purpose is called a recursive message. In other words, it can be said that the recursive message is a special case of the self message as it represents the recursive calls.

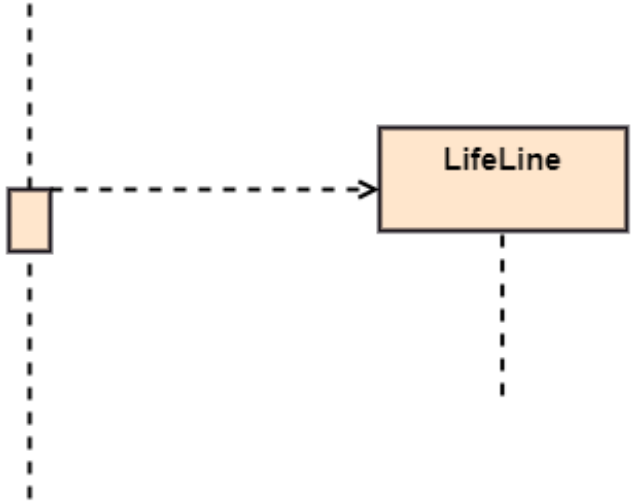


CASE TOOLs – Sequence Diagram

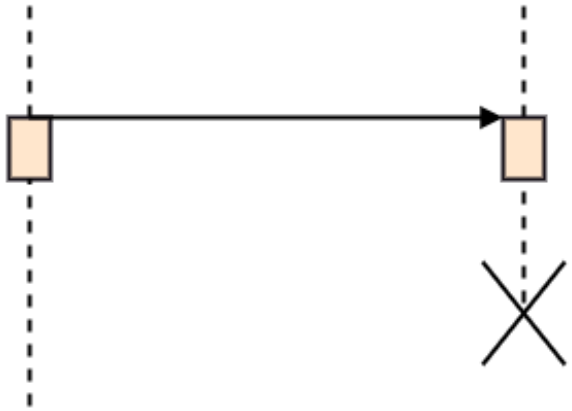
❑Notations of a Sequence Diagram:

❑Messages:

- **Create Message:** It describes a communication, particularly between the lifelines of an interaction describing that the target (lifeline) has been instantiated.



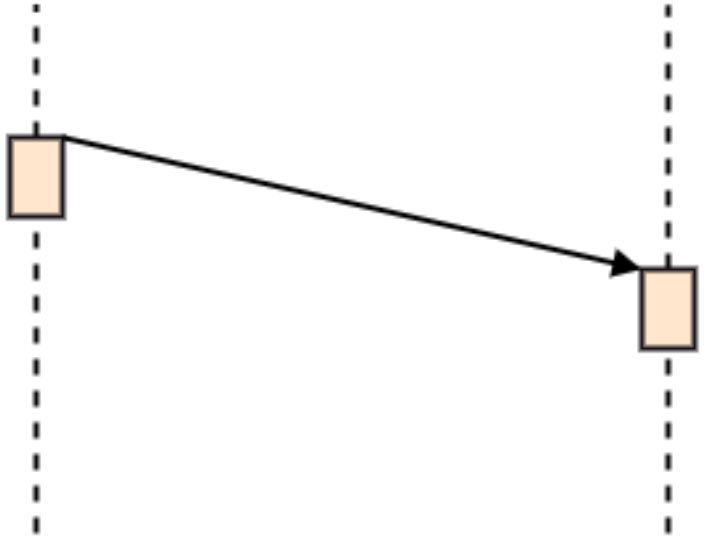
- **Destroy Message:** It describes a communication, particularly between the lifelines of an interaction that depicts a request to destroy the lifecycle of the target.



❑Notations of a Sequence Diagram:

❑Messages:

- **Duration Message:** It describes a communication particularly between the lifelines of an interaction, which portrays the time passage of the message while modeling a system.



CASE TOOLs – Sequence Diagram

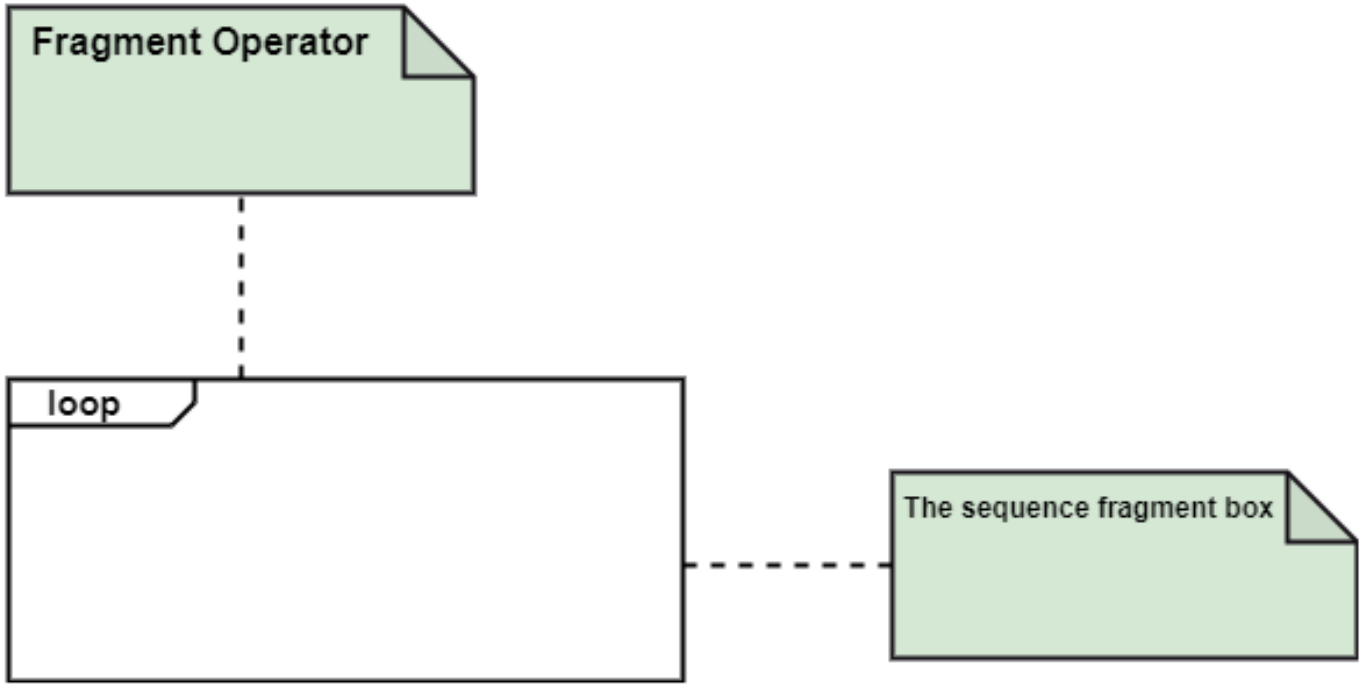
❑Notations of a Sequence Diagram:

❑**Note:** A note is the capability of attaching several remarks to the element. It basically carries useful information for the modelers.

CASE TOOLs – Sequence Diagram

❑Sequence Fragments:

1. Sequence fragments have been introduced by UML 2.0, which makes it quite easy for the creation and maintenance of an accurate sequence diagram.
2. It is represented by a box called a combined fragment, encloses a part of interaction inside a sequence diagram.
3. The type of fragment is shown by a fragment operator.



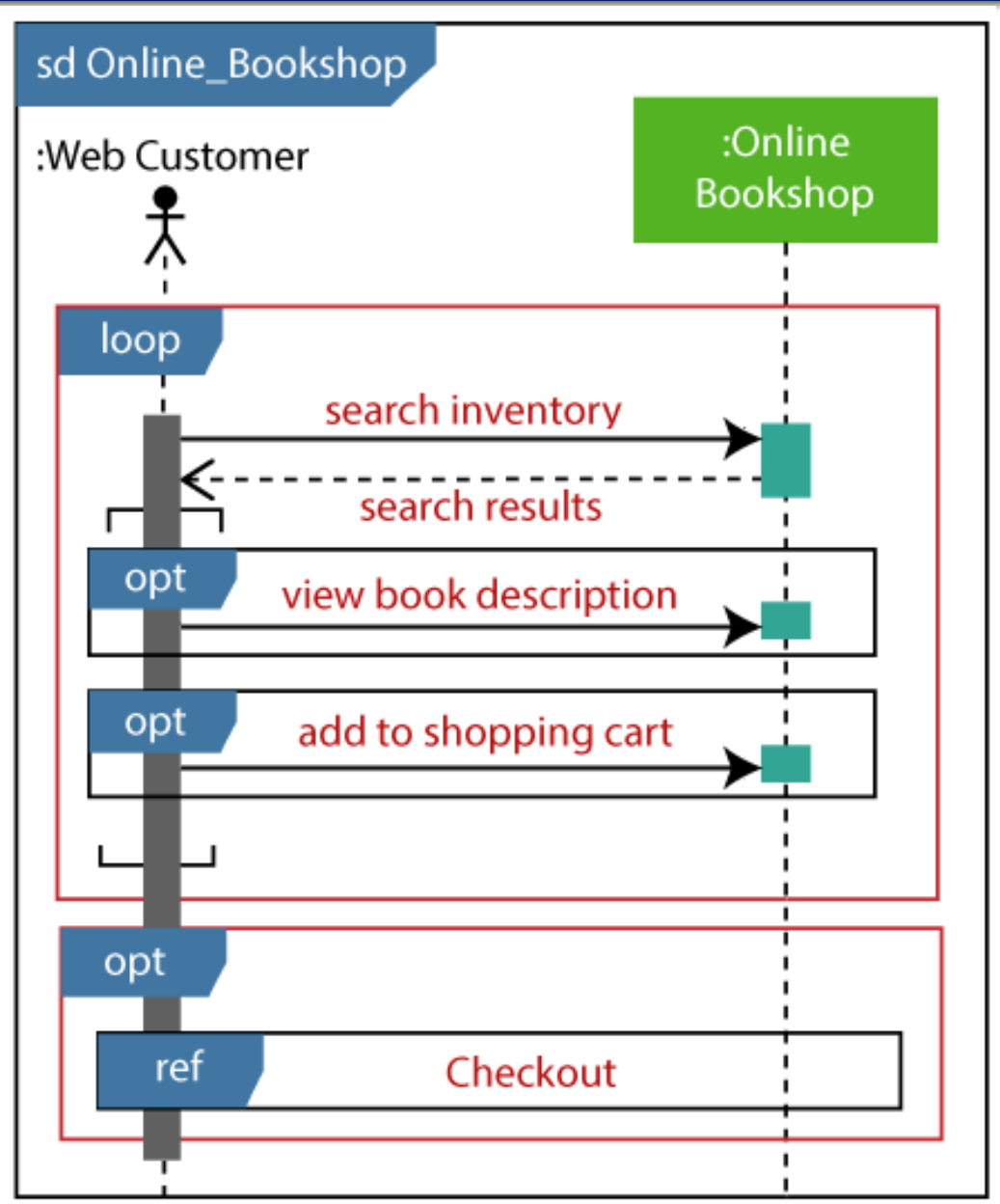
CASE TOOLs – Sequence Diagram

❑Types of Fragments:

Operator	Fragment Type
alt	Alternative multiple fragments: The only fragment for which the condition is true, will execute.
opt	Optional: If the supplied condition is true, only then the fragments will execute. It is similar to alt with only one trace.
par	Parallel: Parallel executes fragments.
loop	Loop: Fragments are run multiple times, and the basis of interaction is shown by the guard.
region	Critical region: Only one thread can execute a fragment at once.
neg	Negative: A worthless communication is shown by the fragment.
ref	Reference: An interaction portrayed in another diagram. In this, a frame is drawn so as to cover the lifelines involved in the communication. The parameter and return value can be explained.
sd	Sequence Diagram: It is used to surround the whole sequence diagram.

CASE TOOLs – Sequence Diagram

- Example of a high-level sequence diagram for online bookshop is given below.
- Any online customer can search for a book catalog, view a description of a particular book, add a book to its shopping cart, and do checkout.



CASE TOOLs – Collaboration Diagram

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

CASE TOOLs – Collaboration Diagram

❑**Notations of a Collaboration Diagram:** Following are the components of a component diagram that are enlisted below:

1. **Objects**
2. **Actors**
3. **Links**
4. **Messages**

CASE TOOLs – Collaboration Diagram

❑Notations of a Collaboration Diagram:

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

❑In the collaboration diagram, objects are utilized in the following ways: The object is represented by specifying their name and class.

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

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

CASE TOOLs – Collaboration Diagram

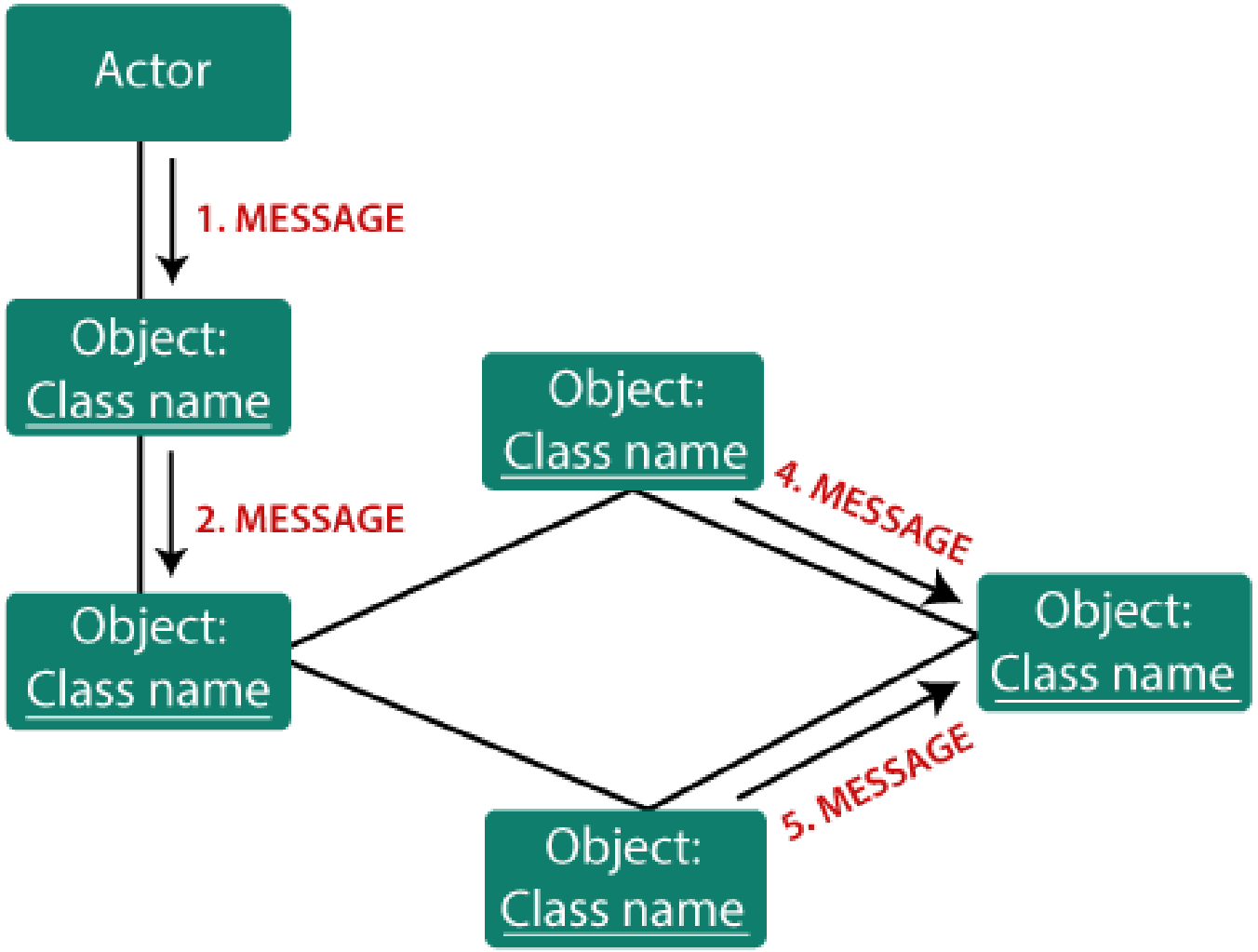
❑Notations of a Collaboration Diagram:

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

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

CASE TOOLs – Collaboration Diagram

❑Components of a Collaboration Diagram:



CASE TOOLs – Collaboration Diagram

❑When to use a Collaboration Diagram?

❑The collaborations are used when it is essential to depict the relationship between the object.

❑Both the sequence and collaboration diagrams represent the same information, but the way of portraying it quite different.

❑The collaboration diagrams are best suited for analyzing use cases.

1. To model collaboration among the objects or roles that carry the functionalities of use cases and operations.
2. To model the mechanism inside the architectural design of the system.
3. To capture the interactions that represent the flow of messages between the objects and the roles inside the collaboration.
4. To model different scenarios within the use case or operation, involving a collaboration of several objects and interactions.
5. To support the identification of objects participating in the use case.
6. In the collaboration diagram, each message constitutes a sequence number, such that the top-level message is marked as one and so on. The messages sent during the same call are denoted with the same decimal prefix, but with different suffixes of 1, 2, etc. as per their occurrence.

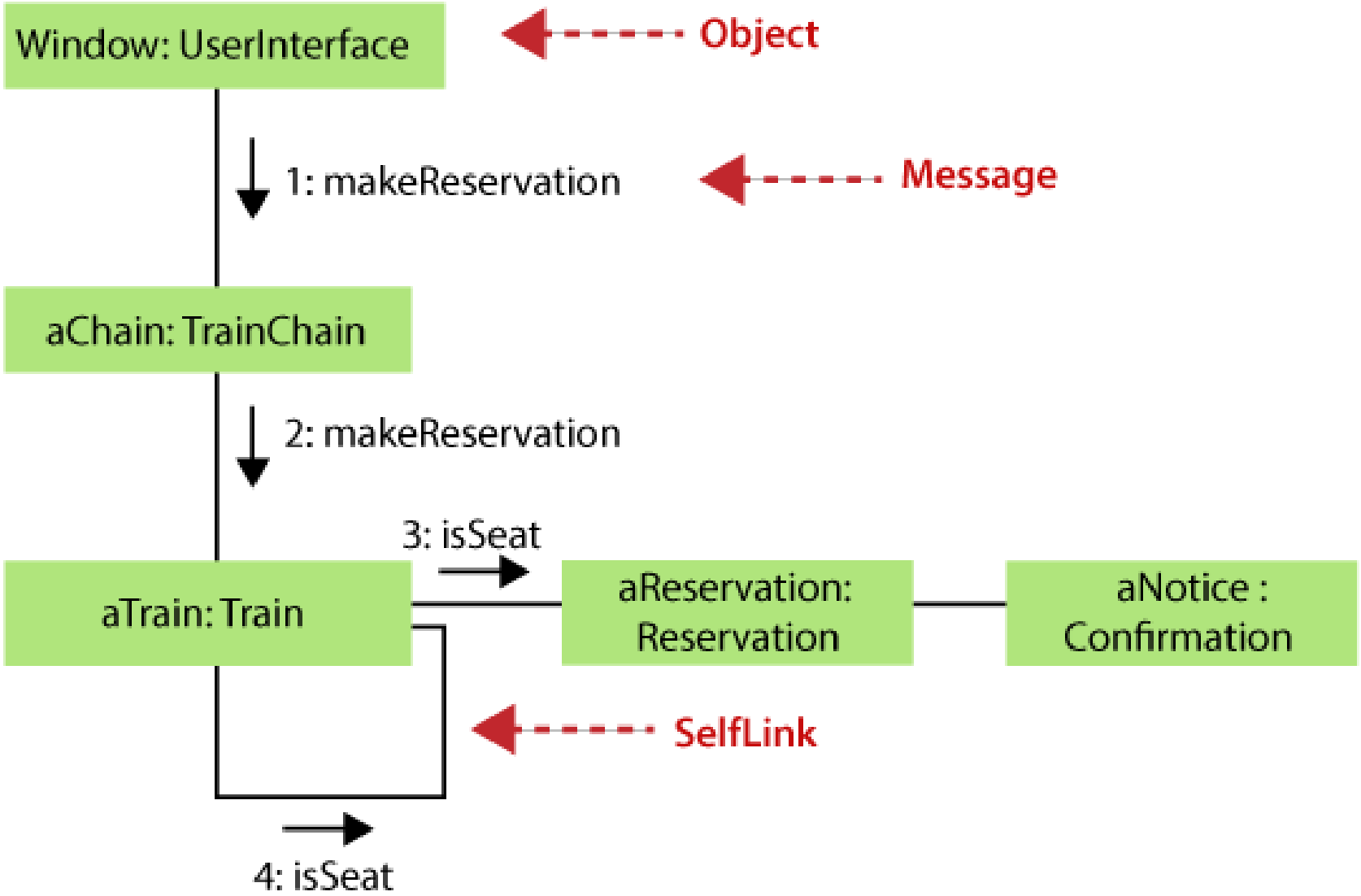
CASE TOOLs – Collaboration Diagram

❑Steps for creating a Collaboration Diagram

1. Determine the behavior for which the realization and implementation are specified.
2. 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.
3. Think through alternative situations that may be involved.
   - Implementation of a collaboration diagram at an instance level, if needed.
   - A specification level diagram may be made in the instance level sequence diagram for summarizing alternative situations.

CASE TOOLs – Collaboration Diagram

❑Example of Collaboration Diagram



CASE TOOLs – State (Machine) Diagram

- The state machine diagram is also called the Statechart or State Transition diagram, which 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.

CASE TOOLs – State (Machine) Diagram

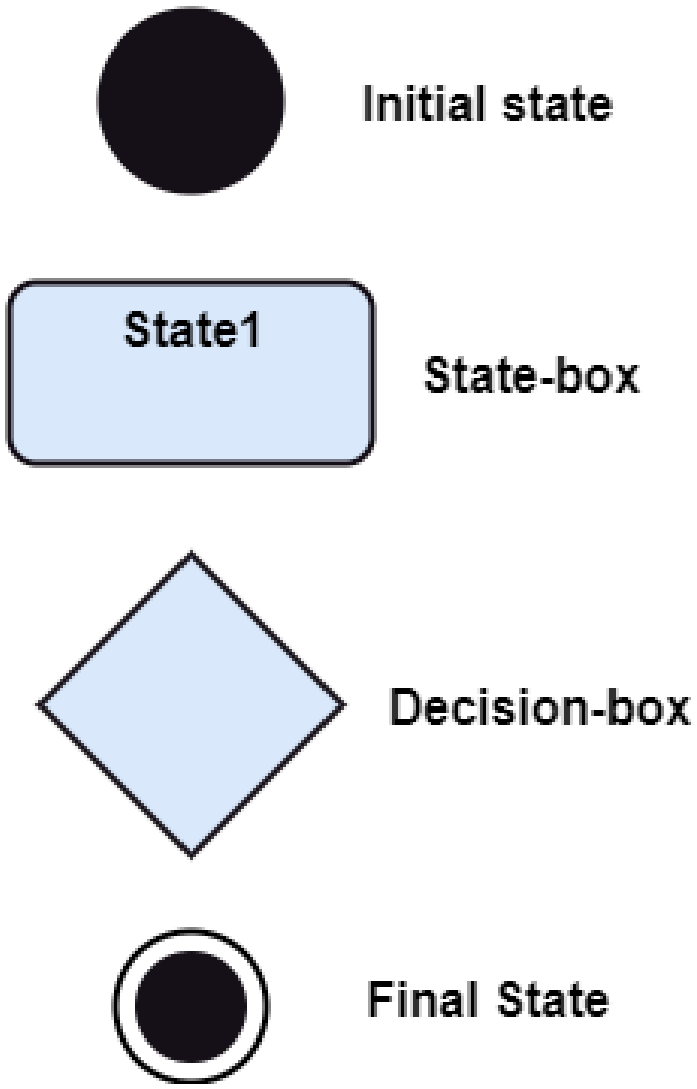
- Following are the types of a state machine diagram that are given below:
- **Behavioral state machine:** The behavioral state machine diagram records the behavior of an object within the system. It depicts an implementation of a particular entity. It models the behavior of the system.
- **Protocol state machine:** It captures the behavior of the protocol. The protocol state machine depicts the change in the state of the protocol and parallel changes within the system. But it does not portray the implementation of a particular component.

CASE TOOLs – State (Machine) Diagram

- **Why State Machine Diagram?**
- Since it records the dynamic view of a system, it portrays the behavior of a software application.
- During a lifespan, an object underwent several states, such that the lifespan exist until the program is executing.
- Each state depicts some useful information about the object.
- 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.
- The main purpose is to depict each state of an individual object.
- It represents an interactive system and the entities inside the system.
- It records the dynamic behavior of the system.

CASE TOOLs – State (Machine) Diagram

❑Following are the notations of a state machine diagram enlisted below:



CASE TOOLs – State (Machine) Diagram

❑**Following are the notations of a state machine diagram enlisted below:**

1. **Initial state:** It defines the initial state (beginning) of a system, and it is represented by a black filled circle.
2. **Final state:** It represents the final state (end) of a system. It is denoted by a filled circle present within a circle.
3. **Decision box:** It is of diamond shape that represents the decisions to be made on the basis of an evaluated guard.
4. **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.
5. **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.

CASE TOOLs – State (Machine) Diagram

❑Types of State: The UML consist of three states:

1. **Simple state:** It does not constitute any substructure.
2. **Composite state:** It consists of nested states (substates), such that it does not contain more than one initial state and one final state. It can be nested to any level.
3. **Submachine state:** The submachine state is semantically identical to the composite state, but it can be reused.

CASE TOOLs – State (Machine) Diagram

❑How to Draw a State Machine Diagram?

- ❑The state machine diagram is used to portray various states underwent by an object.
- ❑The change in one state to another is due to the occurrence of some event.
- ❑All of the possible states of a particular component must be identified before drawing a state machine diagram.
- ❑The primary focus of the state machine diagram is to depict the states of a system.
- ❑These states are essential while drawing a state transition diagram.
- ❑The objects, states, and events due to which the state transition occurs must be acknowledged before the implementation of a state machine diagram.

CASE TOOLs – State (Machine) Diagram

❑How to Draw a State Machine Diagram? Steps

1. A unique and understandable name should be assigned to the state transition that describes the behavior of the system.
2. Out of multiple objects, only the essential objects are implemented.
3. A proper name should be given to the events and the transitions.

CASE TOOLs – State (Machine) Diagram

- State machine diagram is used for:

1. For modeling the object states of a system.
2. For modeling the reactive system as it consists of reactive objects.
3. For pinpointing the events responsible for state transitions.
4. For implementing forward and reverse engineering.

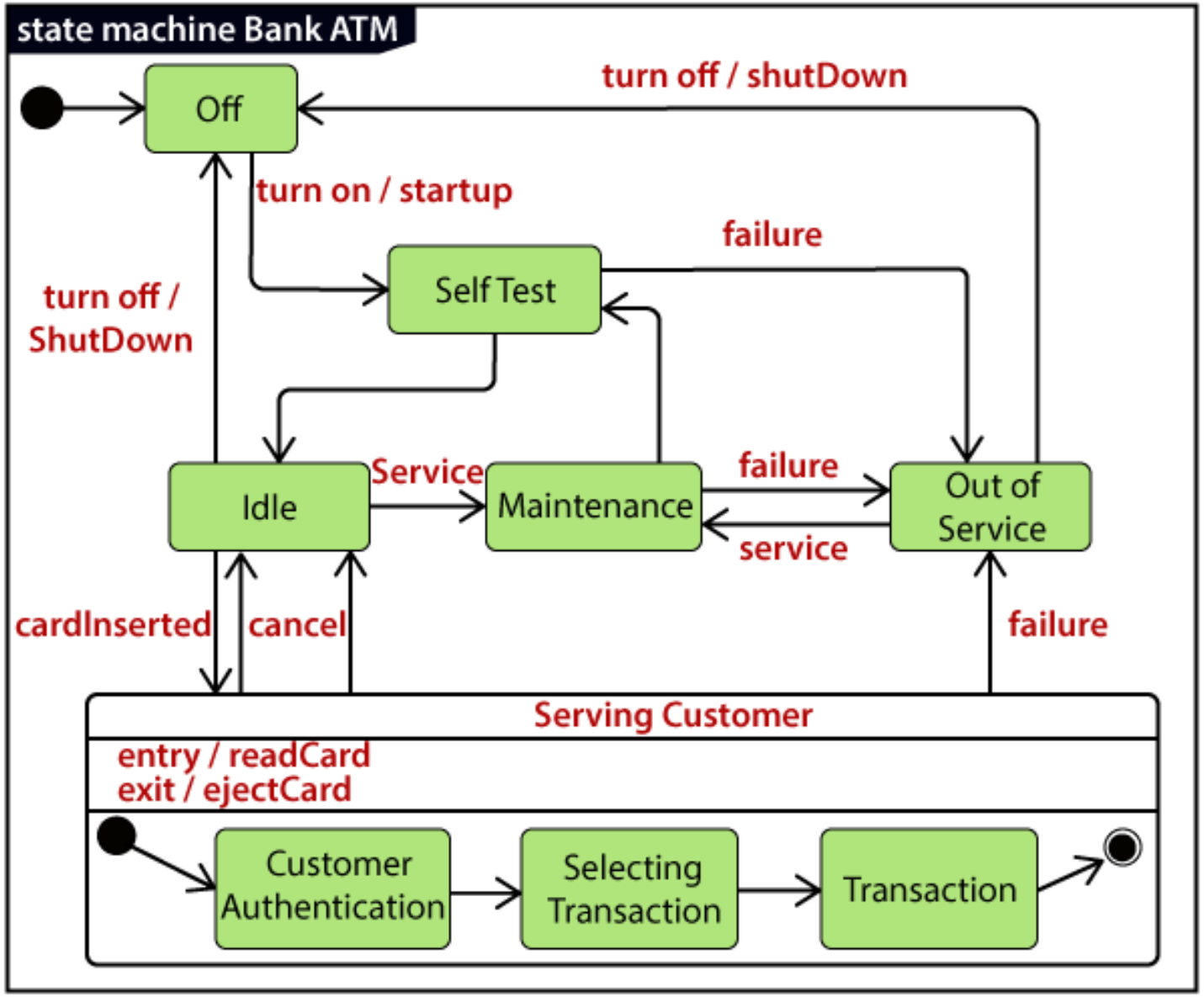
CASE TOOLs – State (Machine) Diagram

❑Example:

1. An example of a top-level state machine diagram showing Bank Automated Teller Machine (ATM) is given below.
2. Initially, the ATM is turned off. After the power supply is turned on, the ATM starts performing the startup action and enters into the **Self Test** state.
3. If the test fails, the ATM will enter into the **Out Of Service** state, or it will undergo **a triggerless transition** to the **Idle** state. This is the state where the customer waits for the interaction.
4. Whenever the customer inserts the bank or credit card in the ATM's card reader, the ATM state changes from **Idle** to **Serving Customer**, the entry action **readCard** is performed after entering into **Serving Customer** state.
5. Since the customer can cancel the transaction at any instant, so the transition from **Serving Customer** state back to the **Idle** state could be triggered by **cancel** event.

CASE TOOLs – State (Machine) Diagram

❑Example:



CASE TOOLs – State (Machine) Diagram

❑Example:

6. Here the **Serving Customer** is a composite state with sequential substates that are **Customer Authentication, Selecting Transaction,** and **Transaction**.
7. **Customer Authentication** and **Transaction** are the composite states itself is displayed by a hidden decomposition indication icon.
8. After the transaction is finished, the **Serving Customer** encompasses a triggerless transition back to the **Idle** state.
9. On leaving the state, it undergoes the exit action **ejectCard** that discharges the customer card.

CASE TOOLs – Activity Diagram

- In the UML, activity diagram is used to demonstrate the flow of control within the system rather than the implementation.
- It models the concurrent and sequential activities.
- The activity diagram helps in envisioning the workflow from one activity to another.
- It put 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.

CASE TOOLs – Activity Diagram

❑Components of an Activity Diagram

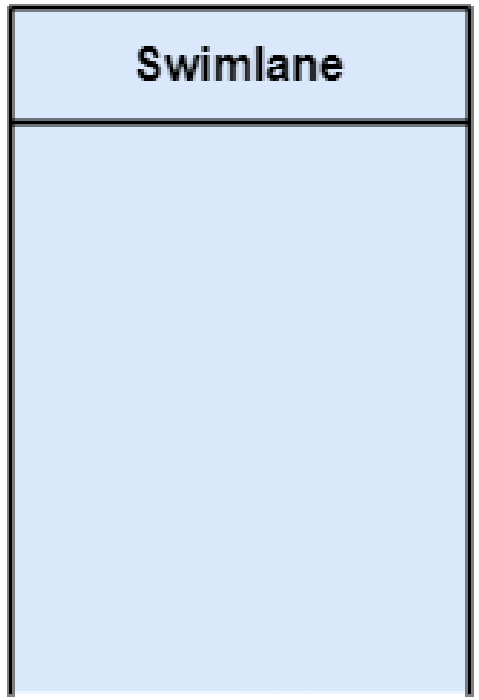
1. Activities :

- The categorization of behavior into one or more actions is termed as an activity.
- In other words, 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.

❑Components of an Activity Diagram

2. Activity partition /swimlane

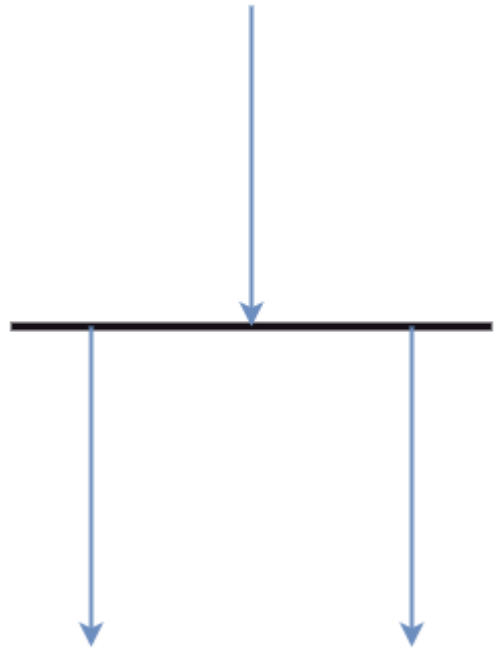
- The swimlane is used to cluster all the related activities in one column or one row.
- It can be either vertical or horizontal. It used to add modularity to the activity diagram.
- It is not necessary to incorporate swimlane in the activity diagram.
- But it is used to add more transparency to the activity diagram.



❑Components of an Activity Diagram

3. Forks

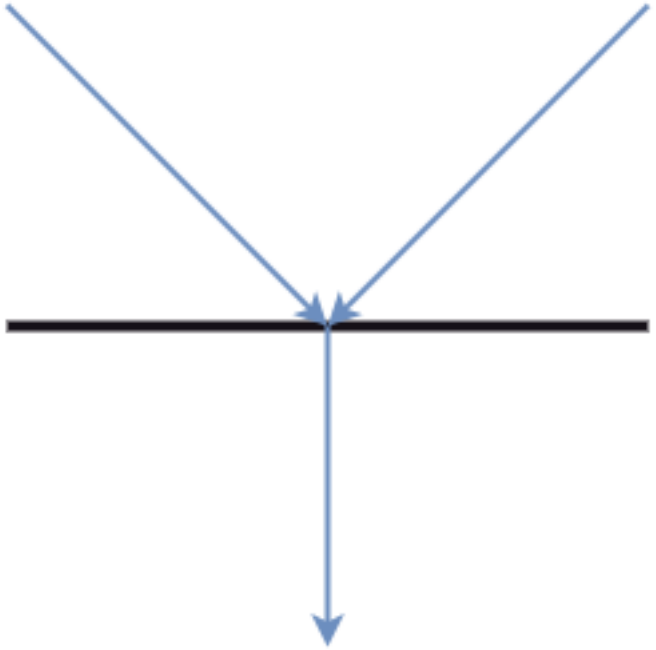
- Forks and join nodes generate the concurrent flow inside the activity.
- A fork node consists of one inward edge and several outward edges.
- It is the same as that of various decision parameters.
- Whenever a data is received at an inward edge, it gets copied and split crossways various outward edges.
- It split a single inward flow into multiple parallel flows.



❑Components of an Activity Diagram

4. Join

- Join nodes are the opposite of fork nodes.
- A Logical AND operation is performed on all of the inward edges as it synchronizes the flow of input across one single output (outward) edge.



CASE TOOLs – Activity Diagram

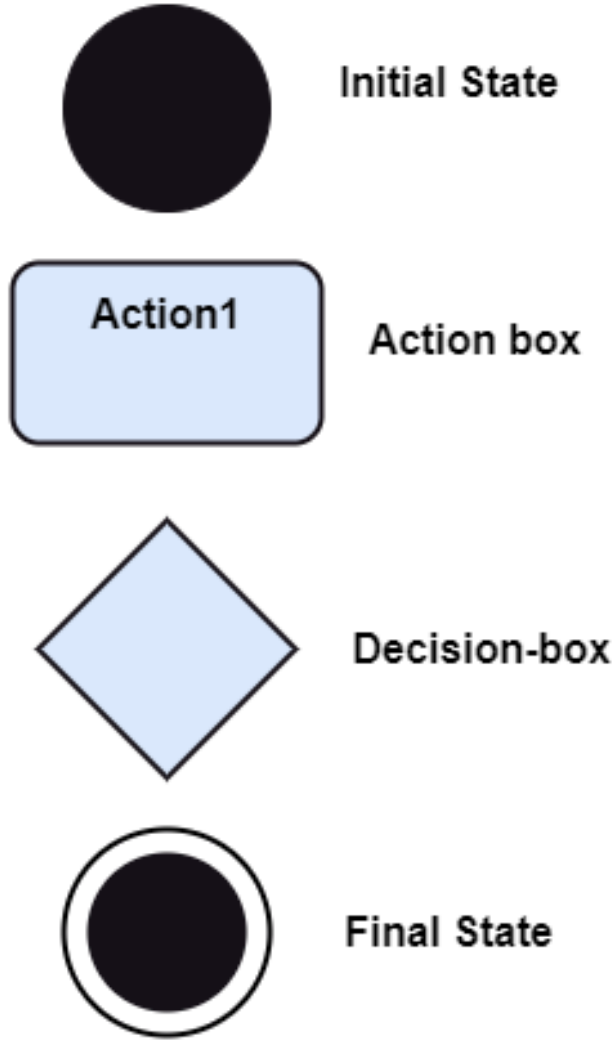
❑Components of an Activity Diagram

5. Pin

- It is a small rectangle, which is attached to the action rectangle.
- It clears out all the messy and complicated thing to manage the execution flow of activities.
- It is an object node that precisely represents one input to or output from the action.

CASE TOOLs – Activity Diagram

- Activity diagram constitutes **following notations**:
- **Initial State:** It depicts the initial stage or beginning of the set of actions.
- **Final State:** It is the stage where all the control flows and object flows end.
- **Decision Box:** It makes sure that the control flow or object flow will follow only one path.
- **Action Box:** It represents the set of actions that are to be performed.



CASE TOOLs – Activity Diagram

❑Why use Activity Diagram?

- ❑An event is created as an activity diagram encompassing a group of nodes associated with edges.
- ❑To model the behavior of activities, they can be attached to any modeling element.
- ❑It can model use cases, classes, interfaces, components, and collaborations.
- ❑It mainly models processes and workflows.
- ❑It envisions the dynamic behavior of the system as well as constructs a runnable system that incorporates forward and reverse engineering.
- ❑It does not include the message part, which means message flow is not represented in an activity diagram.
- ❑It is the same as that of a flowchart but not exactly a flowchart itself. It is used to depict the flow between several activities.

CASE TOOLs – Activity Diagram

❑How to draw an 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.

❑In other words, it can be said that an activity diagram is an enhancement of the flowchart, which encompasses several unique skills.

❑Since it incorporates swimlanes, branching, parallel flows, join nodes, control nodes, and forks, it supports exception handling.

❑A system must be explored as a whole before drawing an activity diagram to provide a clearer view of the user.

❑All of the activities are explored after they are properly analyzed for finding out the constraints applied to the activities.

❑Each and every activity, condition, and association must be recognized.

❑How to draw an Activity Diagram?

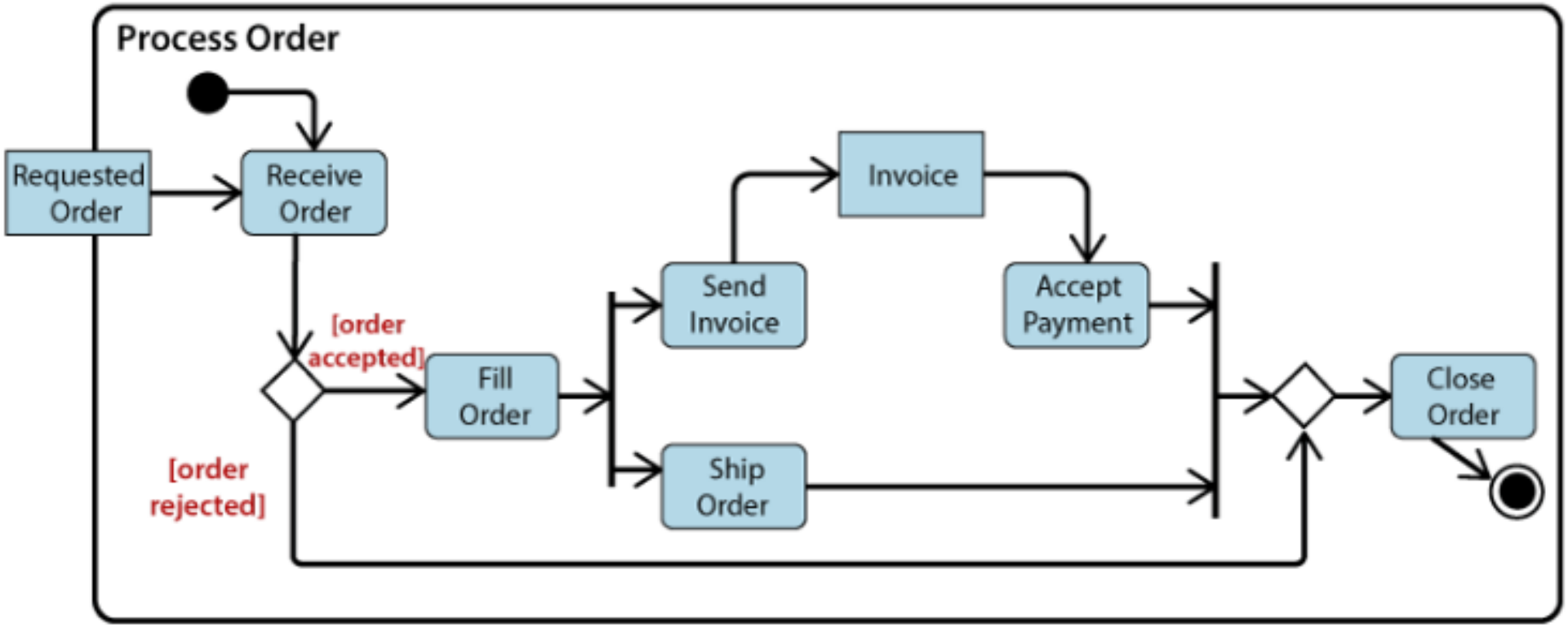
❑After gathering all the essential information, an abstract or a prototype is built, which is then transformed into the actual diagram.

❑Following are the rules that are to be followed for drawing an activity diagram:

1. A meaningful name should be given to each and every activity.
2. Identify all of the constraints.
3. Acknowledge the activity associations.

CASE TOOLs – Activity Diagram

❑**Example:** Here the input parameter is the Requested order, and once the order is accepted, all of the required information is then filled, payment is also accepted, and then the order is shipped. It permits order shipment before an invoice is sent or payment is completed.



CASE TOOLs – Activity Diagram

❑When to use an Activity Diagram?

1. To graphically model the workflow in an easier and understandable way.
2. To model the execution flow among several activities.
3. To model comprehensive information of a function or an algorithm employed within the system.
4. To model the business process and its workflow.
5. To envision the dynamic aspect of a system.
6. To generate the top-level flowcharts for representing the workflow of an application.
7. To represent a high-level view of a distributed or an object-oriented system.

Requirements Management in Agile

❑**Following points describes the requirement management steps followed in different agile software development methodologies.**

❑**Extreme Programming(XP):** Addresses requirements through **user stories** & **onsite customer**. User Stories of two components: written card & conversations after the written card. Written cards are just ”promises for conversation”. Cards need not be complete or clearly stated. Story cards destroyed after implementation.

❑**Scrum:** Also addresses requirements through **user stories**. Thus discussion of user stories which defines actual requirements. So, product owner plays the lead role in the development of the software.

❑**Feature Driven Development(FDD): Gather user requirements & represents in a UML diagram with a list of features**. Feature list manage functional requirements & development tasks. Solution requirements analysis begins with a high level examination of the scope of the system & its context. The team assesses the domain in detail for each modeling area. Small groups composes a model for each domain and present the model for peer review.

Requirements Management in Agile

❑**Lean Software Development: User requirements gathering is done by presenting screens to the end-users & getting their input.** Just in time production ideology applied to recognize specific requirements & environment. At the beginning customer provides the needed input presented in small cards or stories. Developers estimate the time needed for the implementation of each card. Work organization changes into self-pulling system ,each morning during stand-up meetings.

❑**Adaptive Software Development(ASD): Requirements gathering is done in speculative phase**. First, setting the project's mission & objectives, understanding constraints, establishing project organization ,identifying & outlining requirements, making initial scope estimates & identifying key project risks. Project initiation data is gathered in a preliminary JAD sessions.

Requirements Management in Agile

❑**Kanban: User stories help to understand what the actual goals of a sprint were**. A sprint contains one story card. The tasks divides a user story into smaller pieces. A story is divided into client-side & server-side task. The tasks were divided into sub-tasks. Developers minimize the amount of items within a sprint to maintain time of the project.

❑**Agile Unified Process(AUP): Requirement phase includes identifying the stakeholders ,understanding the user's problem, establishing a basis of estimation & defining user interface for the system**. Activities occur during the Inception phase & Elaboration phases but continue through the phases to improve the unfolding design. The deliverables are the business use case model. In construction phase, user stories implemented & iteratively reworked to reflect understanding of problem domain as the project progresses.

Requirements Management in Agile

❑Benefits of agile requirements

1. **Clearly define goals:** Agile requirements identify the most important features or functions of a product. Teams then have a checklist of elements they need to include to define their goals clearly for each project.
2. **Encourage collaboration:** Establishing agile requirements can increase collaboration among team members and ensure everyone is working toward the same goals.
3. **Improve quality and customer satisfaction:** Agile requirements identify what key elements a customer or end user wants from a product. By focusing on the end user's needs and desires throughout product development, teams can improve the product's overall quality and increase customer satisfaction.
4. **Measure success:** Teams that use agile requirements may have an easier time measuring a project's success because they have key performance indicators (KPIs) in place. These KPIs allow team members and clients to identify whether the finished product meets the criteria.

Requirements Management in Agile

Traditional Requirements Management	Agile Requirements Management
In a "waterfall" project management environment, the approach is to capture and define **all** of the end-state requirements upfront.	In an Agile project management environment, while high-level requirements are also captured upfront, it is understood that requirements may evolve over the course of the effort.
Requirements are documented in a business requirements document (BRD) or business specifications document (BSD) for the purpose of designing the end state of a product.	The effort begins with a high-level scope agreement where initial business requirements are translated into user stories. Those user stories specify the needs of the product based on the information at the time given.
The goal is to understand the "as-is" state of the existing product or the business gaps that define the lack, so that the "to-be" state of the desired product can be defined.	The goal is no longer focused on eliciting the "as-is" in order to the define the "to-be," but to clarify and ensure understanding of the business need for all users.
By defining the end-state of the application from the beginning, there is little room for change once development begins.	As more information becomes known over time, the team is better able to adjust and make changes accordingly.

Note for Students

- This power point presentation is for lecture, therefore it is suggested that also utilize the text books and lecture notes.