Chapter 3

Requirement Engineering and Principles

3.1. Introduction to Requirement:

- According to IEEE standard 729, a requirement is defined as follows:

⇒ A condition or capability needed by a user to solve a problem or achieve an objective

⇒ A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification or other formally imposed documents

⇒ A documented representation of a condition or capability, as in 1 and 2.

Types of Software Requirements:

Software Requirements are mainly classified into three types:

- Functional requirements
- Non-functional requirements
- Domain requirements

1. Functional Requirements:

Definition: Functional requirements describe what the software should do. They define the functions or features that the system must have.

Examples:

- **User Authentication**: The system must allow users to log in using a username and password.
- **Search Functionality**: The software should enable users to search for products by name or category.
- **Report Generation**: The system should be able to generate sales reports for a specified date range.

Explanation: Functional requirements specify the actions that the software needs to perform. These are the basic features and functionalities that users expect from the software.

2. Non-functional Requirements:

Definition: Non-functional requirements describe how the software performs a task rather than what it should do. They define the quality attributes, performance criteria, and constraints.

Examples:

- **Performance**: The system should process 1,000 transactions per second.
- **Usability**: The software should be easy to use and have a user-friendly interface.
- **Reliability**: The system must have 99.9% uptime.
- **Security**: Data must be encrypted during transmission and storage.

Explanation: Non-functional requirements are about the system's behaviour, quality, and constraints. They ensure that the software meets certain standards of performance, usability, reliability, and security.

3. Domain Requirements:

Definition: Domain requirements are specific to the domain or industry in which the software operates. They include terminology, rules, and standards relevant to that particular domain.

Examples:

- **Healthcare**: The software must comply with HIPAA regulations for handling patient data.
- **Finance**: The system should adhere to GAAP standards for financial reporting.
- **E-commerce**: The software should support various payment gateways like PayPal, Stripe, and credit cards.

Explanation: Domain requirements reflect the unique needs and constraints of a particular industry. They ensure that the software is relevant and compliant with industry-specific regulations and standards.

Advantages of Classifying Software Requirements:

1. **Better organization:** Classifying software requirements helps organize them into groups that are easier to manage, prioritize, and track throughout the development process.
2. **Improved communication:** Clear classification of requirements makes it easier to communicate them to stakeholders, developers, and other team members. It also ensures that everyone is on the same page about what is required.
3. **Increased quality:** By classifying requirements, potential conflicts or gaps can be identified early in the development process. This reduces the risk of errors, omissions, or misunderstandings, leading to higher-quality software.
4. **Improved traceability:** Classifying requirements helps establish traceability, which is essential for demonstrating compliance with regulatory or quality standards.

Disadvantages of classifying software requirements:

1. **Complexity**: Classifying software requirements can be complex, especially if there are many stakeholders with different needs or requirements. It can also be time-consuming to identify and classify all the requirements.
2. **Rigid structure**: A rigid classification structure may limit the ability to accommodate changes or evolving needs during the development process. It can also lead to a siloed approach that prevents the integration of new ideas or insights.
3. **Misclassification**: Misclassifying requirements can lead to errors or misunderstandings that can be costly to correct later in the development process.

3.2. Requirements Modelling Principles:

⇒ Requirement modelling is a crucial phase in the software development lifecycle, where we translate abstract needs and goals into concrete, tangible representations.

⇒ To ensure effective and efficient modelling, certain key principles should be followed:

1. Abstraction:

- **Focus on essential details:** Identify the core functionalities and characteristics of the system without getting bogged down in implementation specifics.
- **Create high-level views:** Develop models that provide a broad overview of the system, gradually refining them to include more details as needed.

2. Decomposition:

- **Break down complexity:** Divide the system into smaller, more manageable components or modules.
- **Modularize:** Organize these components in a way that promotes independence and reusability.

3. Projection:

- **Consider different perspectives:** Model the system from the viewpoints of various stakeholders, such as end-users, developers, and system administrators.
- **Address diverse concerns:** Ensure that the model captures functional, non-functional, and quality-of-service requirements.

4. Modularity:

- **Promote independence:** Design modules with well-defined interfaces and minimal dependencies on other modules.
- **Enhance maintainability:** Make it easier to understand, modify, and test individual components.

5. Consistency:

- **Avoid contradictions:** Ensure that different parts of the model align and do not conflict with each other.
- **Maintain coherence:** Use a consistent notation and terminology throughout the model.

6. Completeness:

- **Capture all requirements:** Identify and document all relevant functional and non-functional requirements.
- **Address all concerns:** Ensure that the model covers all aspects of the system, including user interactions, data flows, and system behaviour.

7. Correctness:

- **Validate against requirements:** Verify that the model accurately reflects the specified requirements.
- **Identify and resolve inconsistencies:** Detect and eliminate any errors or ambiguities in the model.

8. Traceability:

- **Link requirements to design and code:** Establish clear connections between requirements, design artifacts, and implementation code.
- **Facilitate impact analysis:** Enable efficient identification of the impact of changes on different parts of the system.

9. Understandability:

- **Use clear and concise language:** Employ simple and unambiguous terms and phrases.
- **Employ visual aids:** Utilize diagrams and other visual representations to enhance comprehension.

3.3. Domain Analysis and System Models:

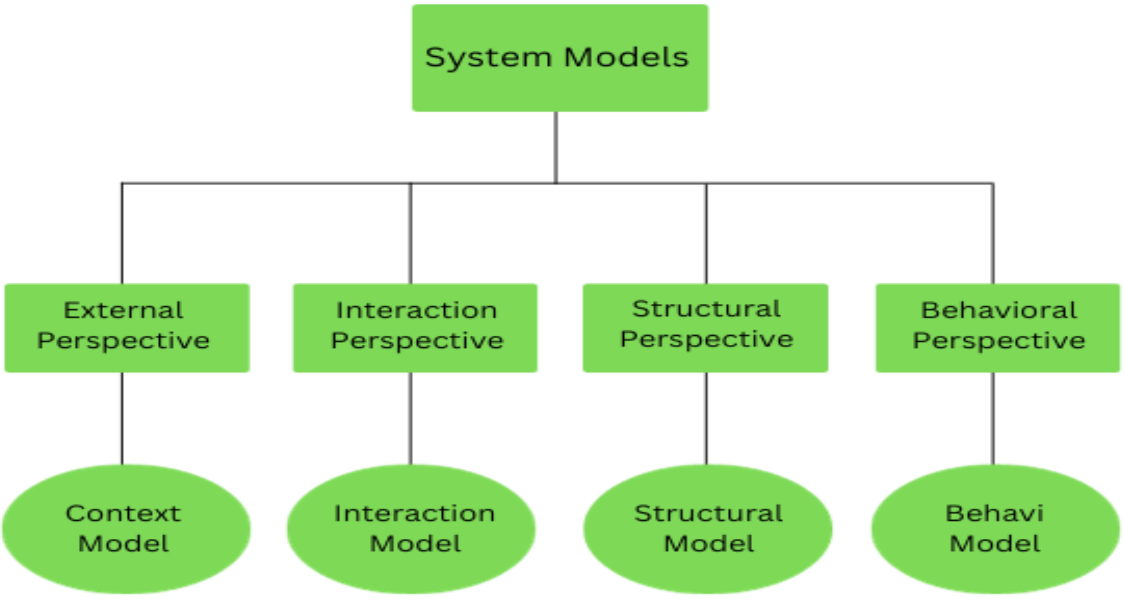
Domain analysis is the process of studying and understanding the business domain, or problem space, where the system will operate. This step is crucial for identifying reusable components, understanding the business rules, and aligning system functionality with user needs. It involves identifying the key concepts, relationships, and rules within that domain. This analysis helps in building a solid foundation for the design and development of software systems.

Key Steps in Domain Analysis:

1. **Identify the Domain:** Clearly define the boundaries of the domain, including its scope and objectives.
2. **Identify Key Concepts:** Discover the essential entities and their attributes within the domain.
3. **Define Relationships:** Determine how these entities interact and relate to each other.
4. **Identify Rules and Constraints:** Establish the rules and regulations that govern the domain.
5. **Create a Domain Model:** Visualize the domain using appropriate modeling techniques, such as UML class diagrams or entity-relationship diagrams.

System Models:

System models are representations of a system's structure, behavior, and interactions with its environment. They help in understanding the system's requirements, design, and implementation.



There are four types of system models, which are as follows:

1. Context Model:

The external perspective model is a crucial step in developing software. The model shows how the system whose abstract view is being created is placed in an environment with the other system. In this step, software developers and stakeholders work together to decide the functionality to be included in the system.

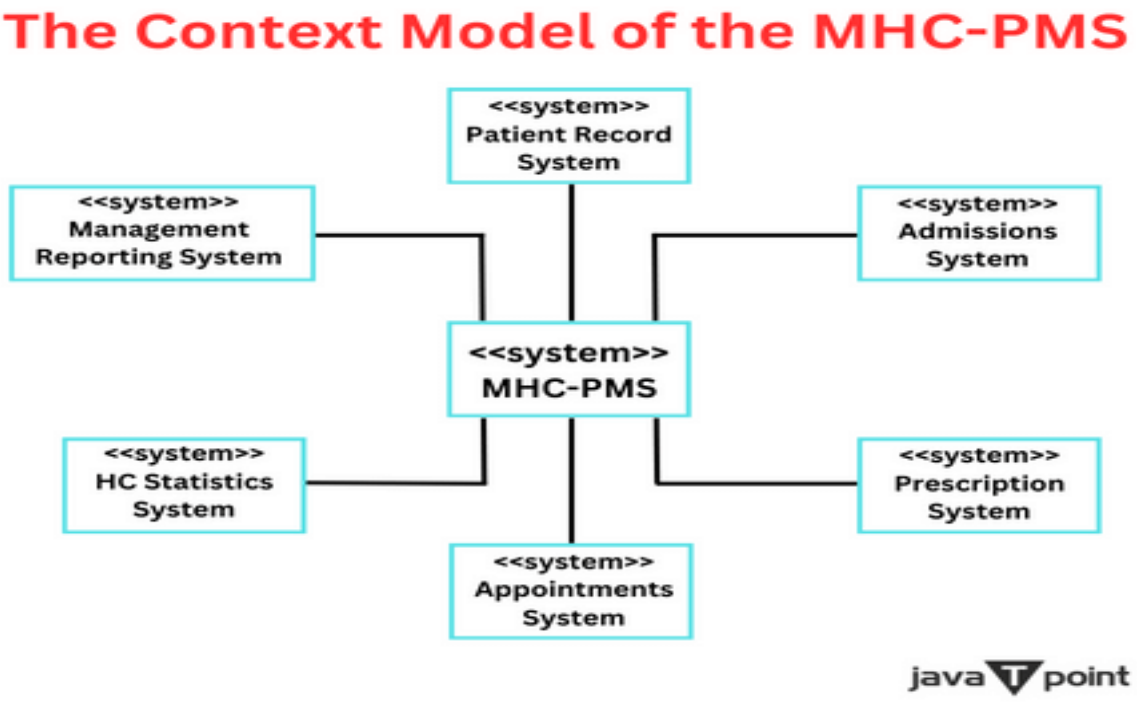
Whenever you use the system, there are so many surrounding systems that we need to deal with as their support is required, which means no software system is used in an isolated manner. It defines what lies outside the boundaries of the system. System boundaries are established to specify the outside and inside of the system. The position of the system boundary affects the system's needs.

The purpose of the context model is to understand the environment in which the system will work, along with the support of the surrounding systems. Software developers develop various architectural models illustrating the system and its association with other systems.

In simple words, we can say that the context model defines how the system interacts with its environment.

Example of context model:

The diagram below shows the main system at the centre, **MHC-PMS (Mental Health Care Patient Management System)**. This system can be used in clinics and help them maintain patients' records. It comprises various sub-systems: the **Patient Record System, Admission System, Prescription System, Appointments System, HC Statistics System,** and **Management Reporting System**. These sub-systems are connected to the main system. These sub-systems support and work with the main system.



2. **Interaction Model:**

The interaction perspective model explains how components of the system interact with each other. There are three types of interactions:

- **User interaction:** It involves user input and user output. It interacts with the user and helps to identify user requirements.
- **System interaction:** It is the interaction between the system which is to be developed and other systems. It interacts from system to system and highlights the problems that may arise in communication.
- **Component interaction:** It interacts with different components of the same system and helps understand whether the proposed system can provide the required system performance.

There are two kinds of diagrams that come under interaction models, which are as follows:

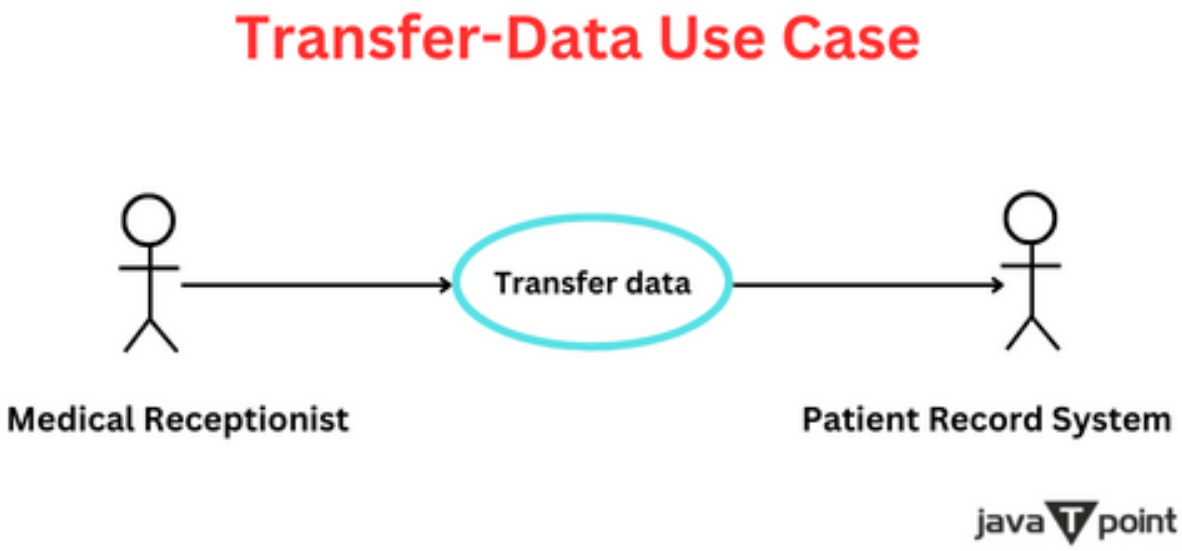
1. Use Case Diagram:

It was presented by Ivar Jacobson in 1987 in the article on use cases. It models interactions between a system and an external user or other system. It is a popular model used to support requirement elicitation and incorporated into the **UML (Unified Modeling Language)**.

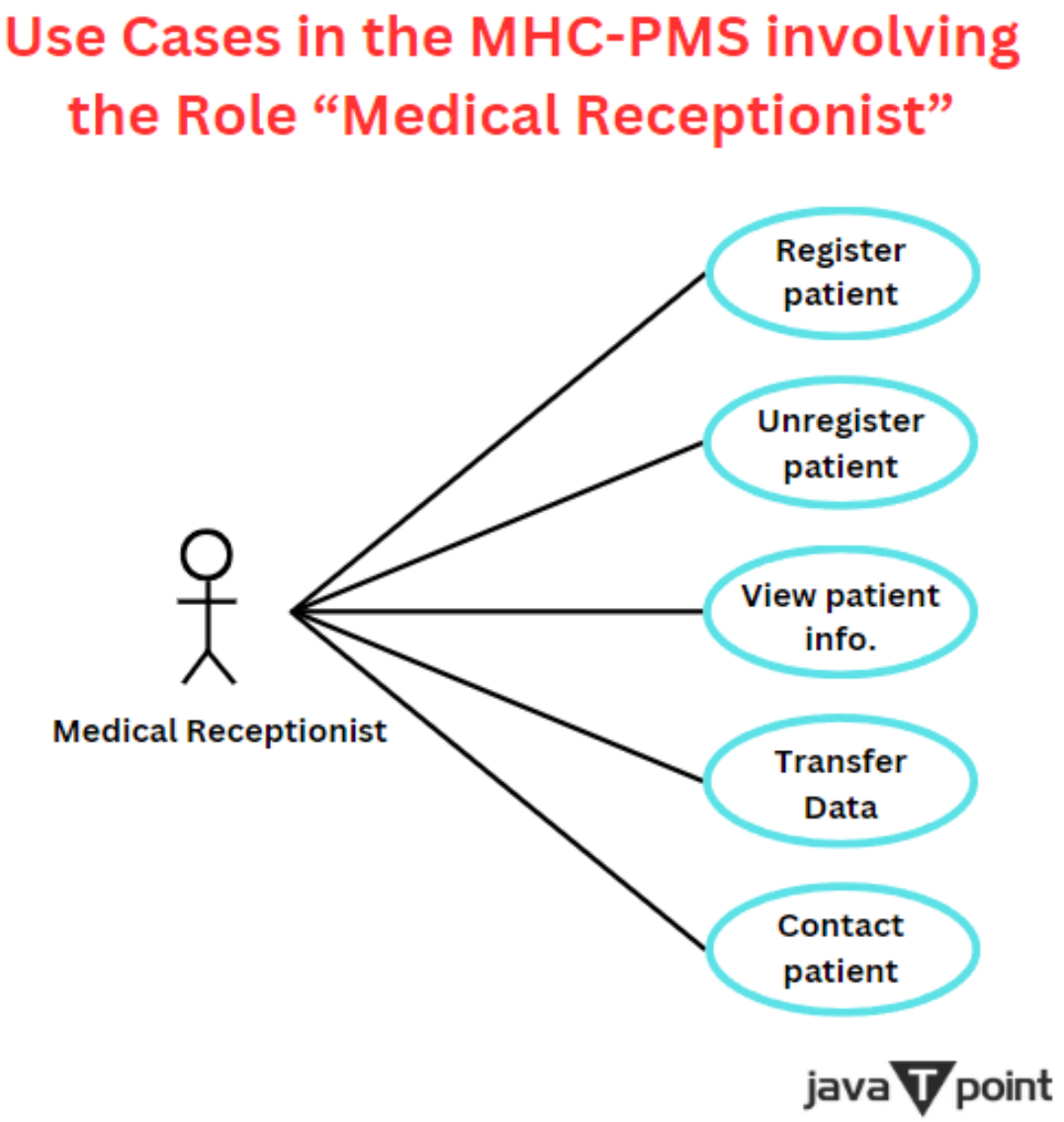
It is a simple scenario to explain the user's needs from a system. Every use case shows a discrete task, including external system interaction.

Example of the Use Case Diagram:

In the simplest form, a **"transfer-data" use case** is shown below:



Composite Use Case Diagram:



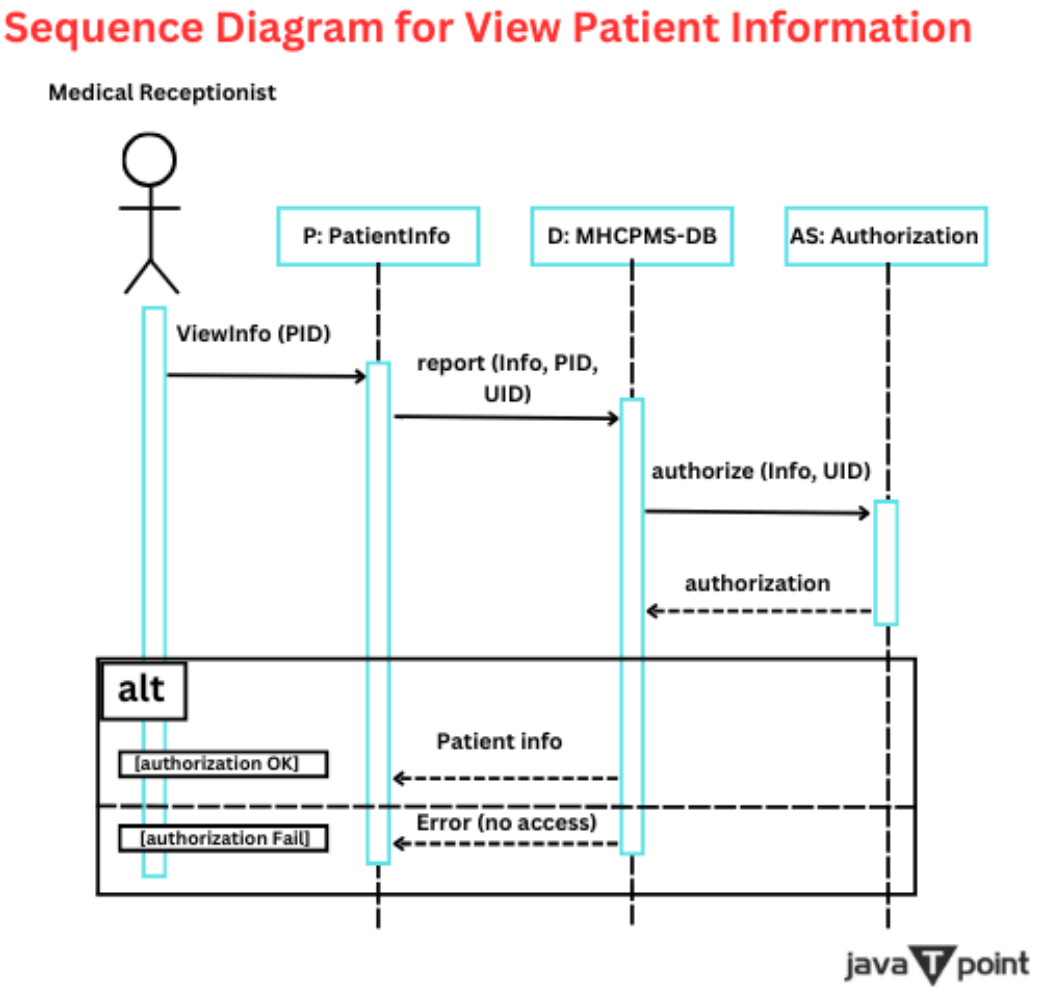
2. Sequence Diagrams:

It is used to model interactions between system components, and external systems may also be involved. This diagram is useful in exhibiting the interaction between the actors and the objects within the system. It represents the sequence of interactions that take place at the time of a certain use case.

Example of Sequence Diagram:

You can see a sequence diagram below that is drawn to view patient information. This example has been taken from MHC-PMS, where viewing patient information is a type of functionality that MHC-PMS provides.

We will see how the sequence diagram is drawn for viewing patient information, which means we will see the different sequences of interactions that will take place.



3. **Structural Model:**

The structural perspective model represents a system's organization in terms of the parts that build the system and their relationships. Structural models can be static models or dynamic models. The static models represent the structure of the system design, and the dynamic models represent the system's organization during execution. When we want to design the system architecture, then the structural model of the system is created.

The structural model is of three types, which are as follows:

1. Class diagrams
2. Generalization
3. Aggregation

We will understand these types one by one.

1. Class diagrams

It is a popular diagram that comes under **UML**. When creating an object-oriented system model, classes and their association are shown in the system, then class diagrams are used. An association is a relation between the classes that connect classes.

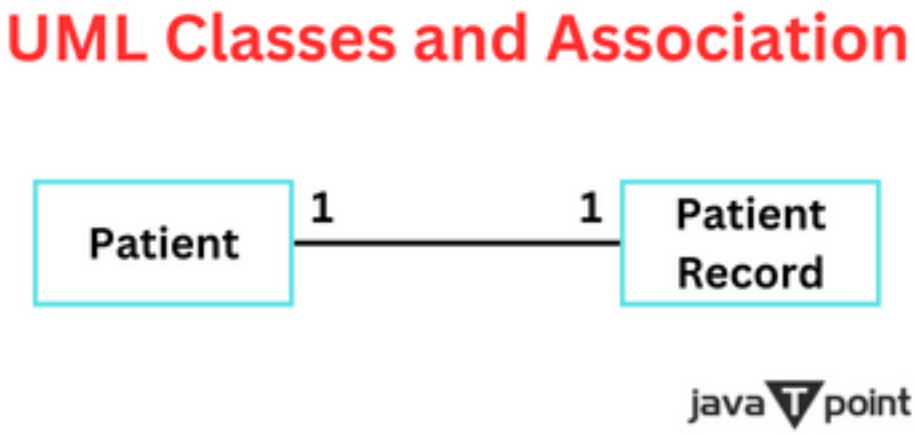
Class diagrams are described at varying levels of detail in UML. An object class is a type of system object.

UML classes and association:

Let us understand how a class diagram is created.

The class name is written inside the rectangular box, and the association between the two classes is shown with the help of a solid line. The feature of a class diagram is the ability to demonstrate the number of objects included in the association.

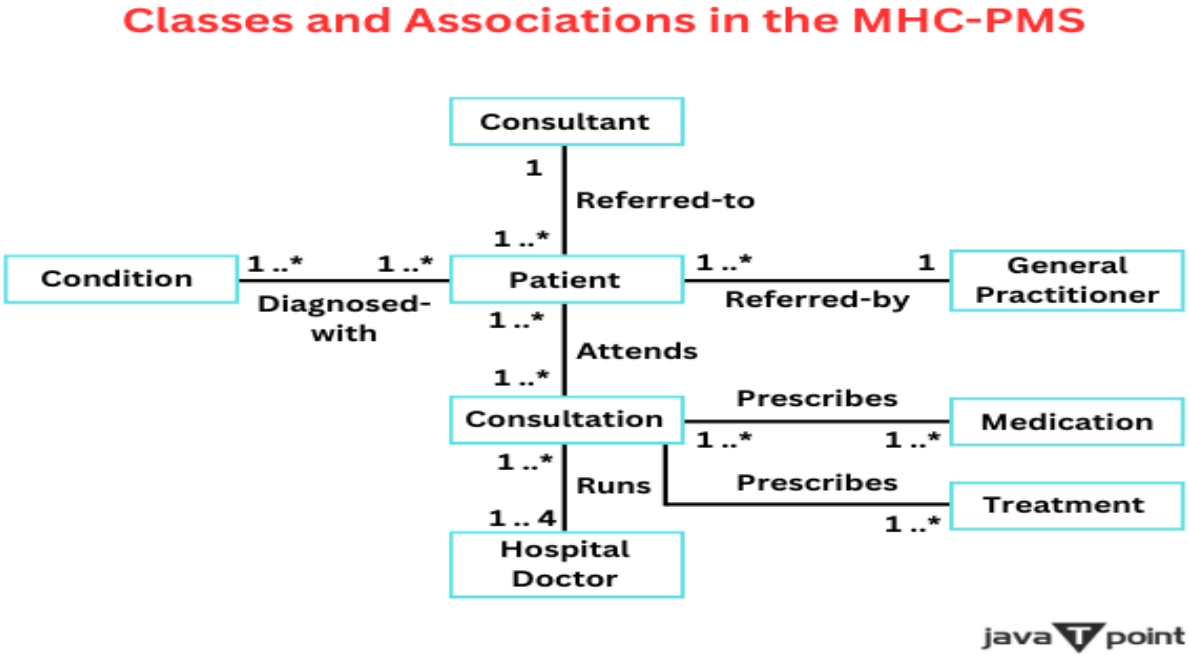
You can see a simple diagram of patient and patient records below:



In the class diagram, the association between patient and patient records is one-to-one. Each end of the association is annotated with 1, which means one patient can have exactly one record.

Classes and associations in the MHC-PMS

In the diagram below, various classes are written inside the rectangular boxes.

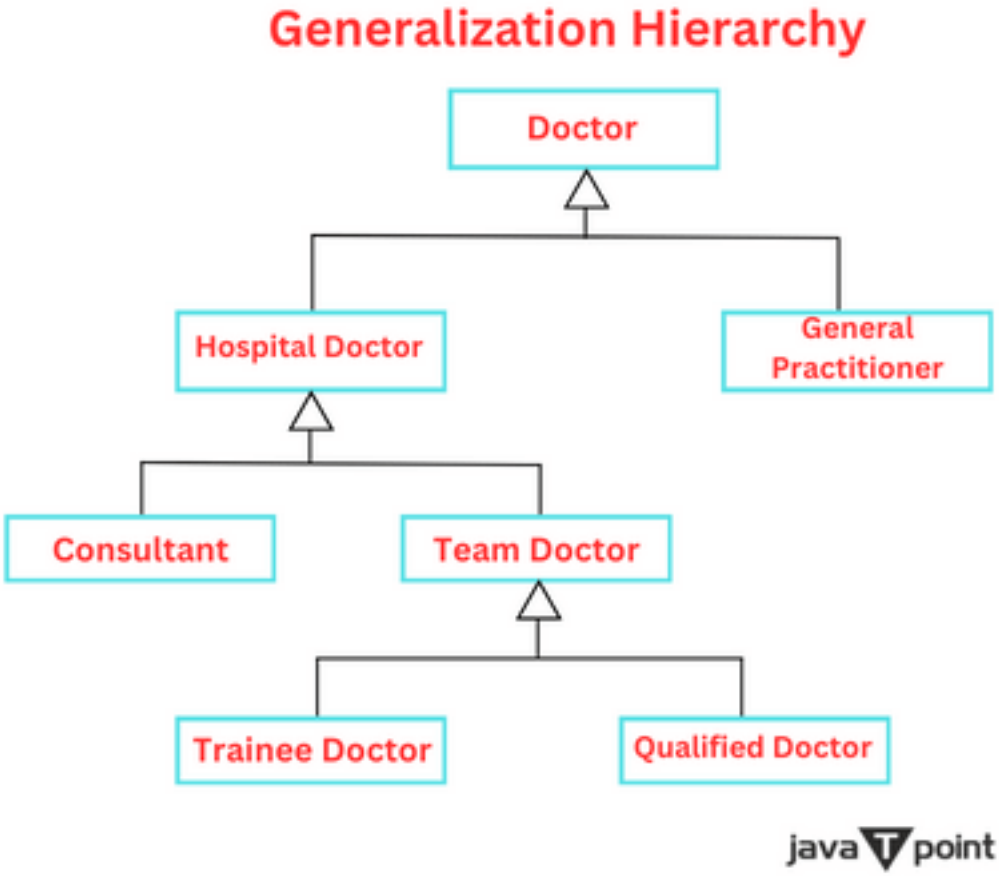


2. Generalization

It is the method that is used to manage the complexity. The entities are placed in more general classes for easy understanding. In object-oriented languages like Java, generalization is implemented with the help of class inheritance mechanisms built into the language.

In generalization, the operations and attributes associated with higher-level classes are also associated with lower-level classes, which are subclasses that inherit the properties from parent classes.

Example of Generalization Hierarchy:



4. **Behavioural Model:**

It is the behavioural perspective model that represents the dynamic behaviour of the system. There are two types of behavioural models:

1. Data-driven modeling
2. Event-driven modeling

Let us comprehend both types in detail one by one.

1. Data-driven modeling

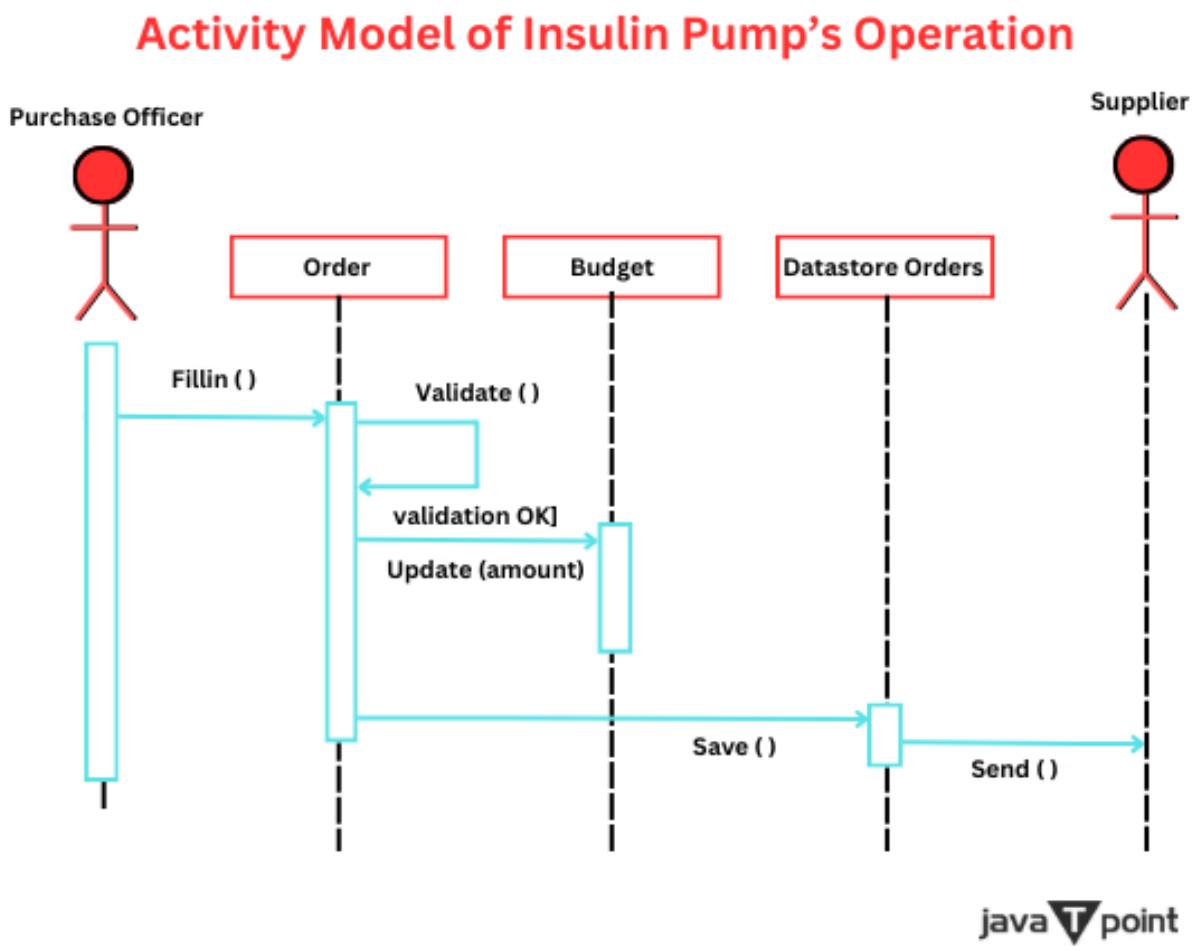
It means data that comes in has to be processed by the system. Data-driven modes are the first graphical software models. Data-driven models represent the actions, which include processing

input data and generating an associated output. These models are used to show end-to-end processing in the system, so they are helpful for analysis of requirements.

Data Flow Diagrams (DFDs) are simple diagrams for tracking and documentation purposes. It helps to understand the developers of the system. It shows the data exchange between a system and other systems within its environment.

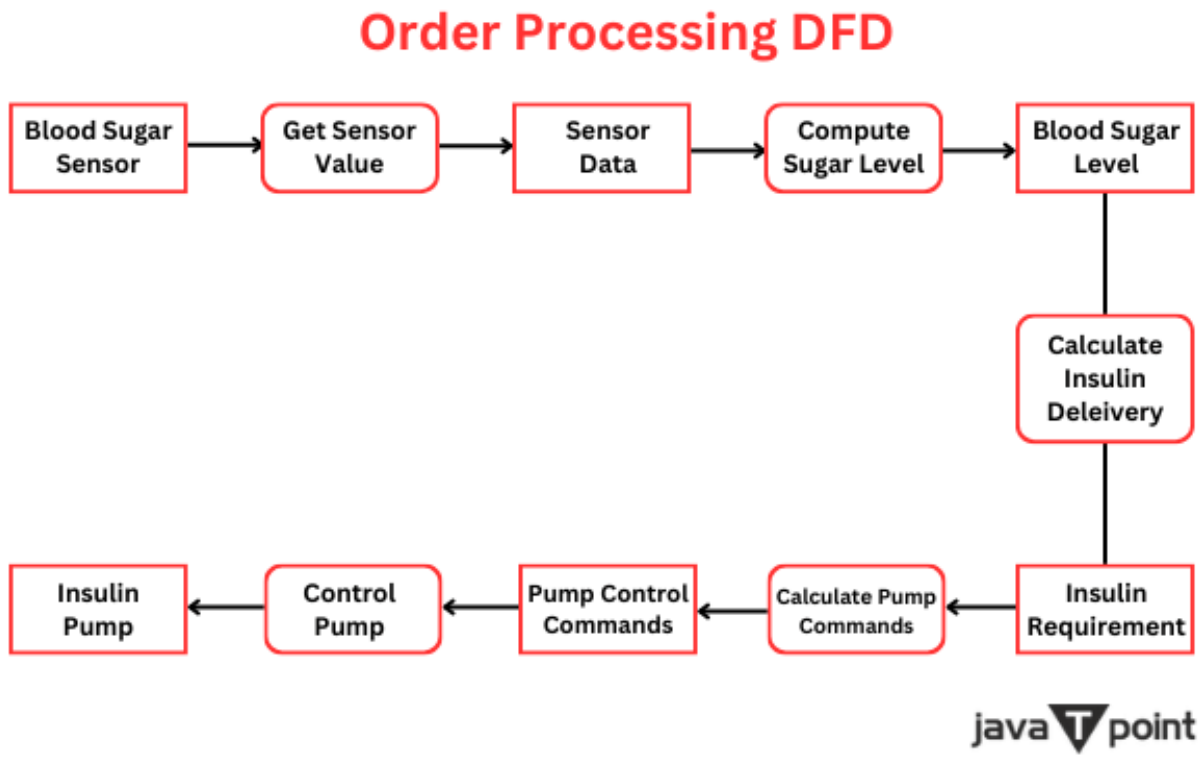
Example of an activity model of an insulin pump's operation:

The diagram below shows an activity model of the insulin pump's operation. You can see the processing steps in the diagram below. The data flowing between these steps is represented as objects. You already know that it is a sequence diagram used to model interaction.



Example for order processing DFD for insulin pump:

The order processing DFD for the insulin pump is shown in the diagram below. In this example, notations used in this model are the rounded boxes for representing the functional boxes, rectangular boxes for representing the data stored, and arrows for representing the data flow between the functions.



Blood Sugar is taken in this example and then analyzed through the sensor. After that, according to the blood sugar, the required insulin computation is done, and the proper amount of insulin is given to the patient to control the sugar level. The data flows in a systematic way to achieve the goal.

2. Event-driven modeling

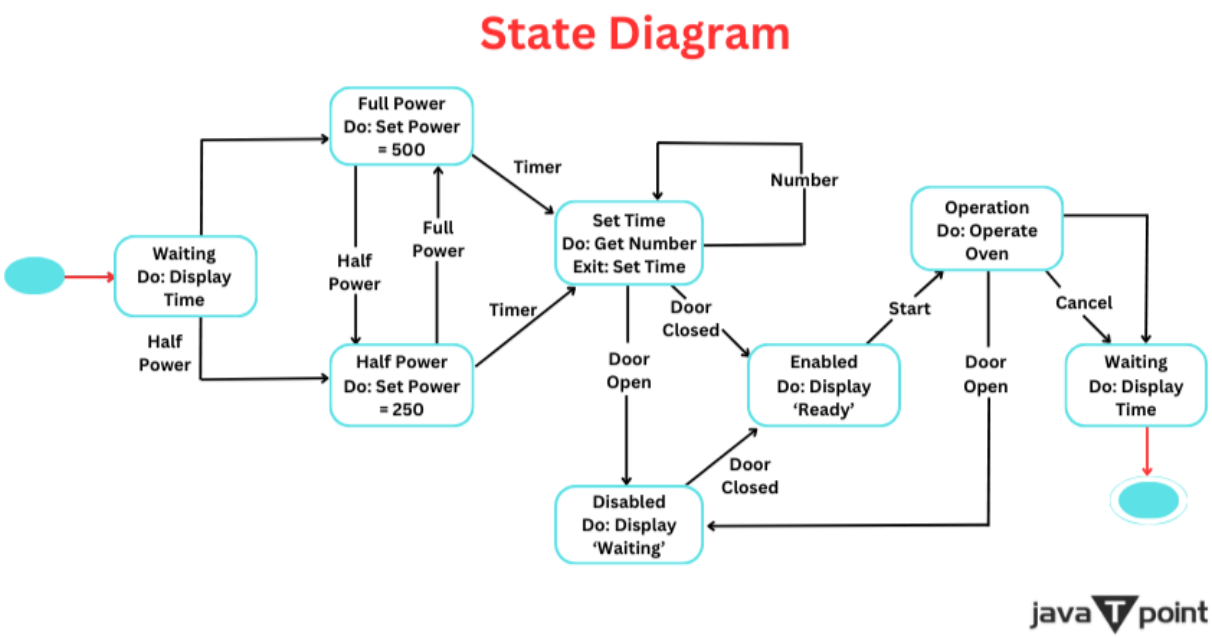
It means that an event occurs that triggers the system, and that event may have associated data. This model represents the response of the system to external and internal events.

This modeling assumes a system has several states, and events can cause transitions between states. It is used to depict the state machine model.

The state machine model represents system states as nodes and events as arcs between these nodes. Event-driven modeling is represented with the help of state diagrams, which are based on state charts.

Example of State Diagram:

You can see a state diagram of a microwave oven below, but real microwaves are much more complex than this state diagram.

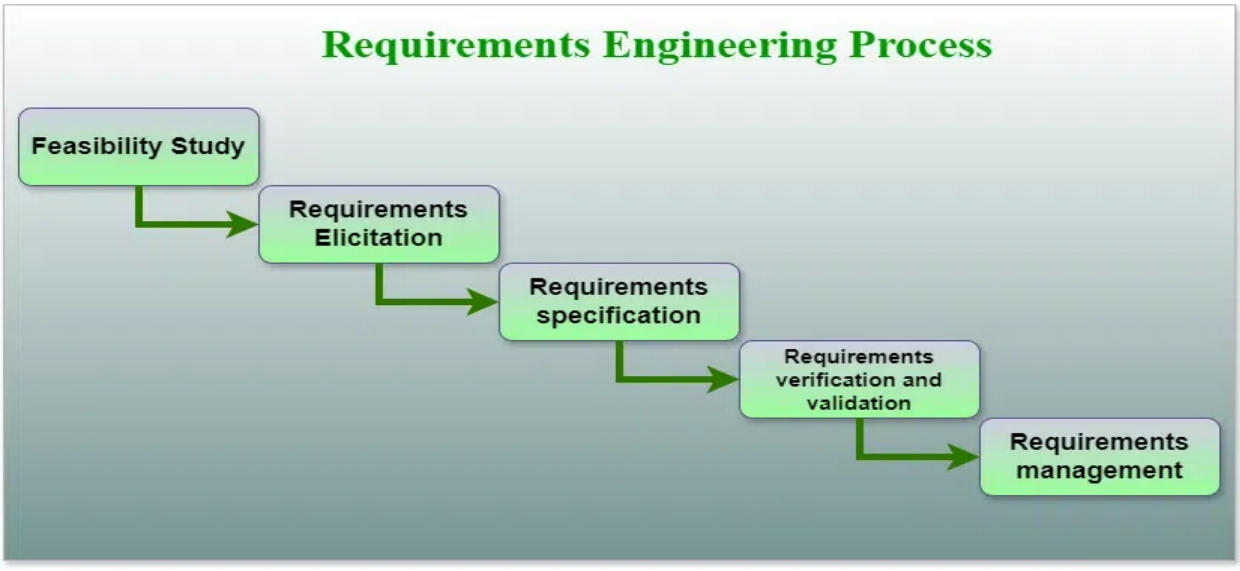


In this example, a simple microwave is used to understand the state diagram easily. A microwave has a switch for selecting half or full power, a numeric keypad for inputting the cooking time, an alphanumeric display, and a start-stop button.

3.4. Requirement 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.

Requirements Engineering Process



1. **Feasibility Study**
2. **Requirements elicitation**
3. **Requirements specification**
4. **Requirements for verification and validation**
5. **Requirements management**

1. <u>Feasibility Study:</u>

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

2. **Technical Feasibility**: In Technical Feasibility current resources both hardware software along required technology are analyzed/assessed to develop the project.
3. **Operational Feasibility:** In Operational Feasibility degree of providing service to requirements is analyzed along with how easy the product will be to operate and maintain after deployment.
4. **Economic Feasibility:** In the Economic Feasibility study cost and benefit of the project are analyzed.
5. **Legal Feasibility:** In legal feasibility, the project is ensured to comply with all relevant laws, regulations, and standards.
6. **Schedule Feasibility:** In schedule feasibility, the project timeline is evaluated to determine if it is realistic and achievable.

2. <u>Requirements Elicitation:</u>

- It is related to the various ways used to gain knowledge about the project domain and requirements.
- Requirements elicitation is the process of gathering information about the needs and expectations of stakeholders for a software system.
- The goal of this step is to understand the problem that the software system is intended to solve and the needs and expectations of the stakeholders who will use the system.

Several techniques can be used to elicit requirements, including:

- **Interviews**: These are one-on-one conversations with stakeholders to gather information about their needs and expectations.
- **Surveys**: These are questionnaires that are distributed to stakeholders to gather information about their needs and expectations.
- **Focus Groups**: These are small groups of stakeholders who are brought together to discuss their needs and expectations for the software system.
- **Observation**: This technique involves observing the stakeholders in their work environment to gather information about their needs and expectations.
- **Prototyping**: This technique involves creating a working model of the software system, which can be used to gather feedback from stakeholders and to validate requirements.

It's important to document, organize, and prioritize the requirements obtained from all these techniques to ensure that they are complete, consistent, and accurate.

3. Requirements Specification:

- Requirements specification is the process of documenting the requirements identified in the analysis step in a clear, consistent, and unambiguous manner.
- The goal of this step is to create a clear and comprehensive document that describes the requirements for the software system.
- This document should be understandable by both the development team and the stakeholders.

Several types of requirements are commonly specified in this step, including

1. **Functional Requirements:** These describe what the software system should do. They specify the functionality that the system must provide, such as input validation, data storage, and user interface.
2. **Non-Functional Requirements:** These describe how well the software system should do it. They specify the quality attributes of the system, such as performance, reliability, usability, and security.
3. **Constraints:** These describe any limitations or restrictions that must be considered when developing the software system.
4. **Acceptance Criteria**: These describe the conditions that must be met for the software system to be considered complete and ready for release.

4. Requirements Verification and Validation:

- **Verification:** It refers to the set of tasks that ensures that the software correctly implements a specific function.
- **Validation:** It refers to a different set of tasks that ensures that the software that has been built is traceable to customer requirements. I
- f requirements are not validated, errors in the requirement definitions would propagate to the successive stages resulting in a lot of modification and rework.

The main steps for this process include:

1. The requirements should be consistent with all the other requirements i.e. no two requirements should conflict with each other.
2. The requirements should be complete in every sense.
3. The requirements should be practically achievable.
   - The goal of V&V is to ensure that the software system being developed meets the requirements and that it is developed on time, within budget, and to the required quality.
   - Verification and Validation is an iterative process that occurs throughout the software development life cycle.
   - It is important to involve stakeholders and the development team in the V&V process to ensure that the requirements are thoroughly reviewed and tested.

5. Requirements Management:

- Requirements management is the process of managing the requirements throughout the software development life cycle, including tracking and controlling changes, and ensuring that the requirements are still valid and relevant.
- The goal of requirements management is to ensure that the software system being developed meets the needs and expectations of the stakeholders and that it is developed on time, within budget, and to the required quality.

Several key activities are involved in requirements management, including:

1. **Tracking and controlling changes:** This involves monitoring and controlling changes to the requirements throughout the development process, including identifying the source of the change, assessing the impact of the change, and approving or rejecting the change.
2. **Version control**: This involves keeping track of different versions of the requirements document and other related artifacts.
3. **Traceability**: This involves linking the requirements to other elements of the development process, such as design, testing, and validation.
4. **Communication:** This involves ensuring that the requirements are communicated effectively to all stakeholders and that any changes or issues are addressed promptly.
5. **Monitoring and reporting**: This involves monitoring the progress of the development process and reporting on the status of the requirements.

Tools Involved in Requirement Engineering:

- Observation report
- Questionnaire (survey, poll)
- Use cases
- User stories
- Requirement workshop
- Mind mapping
- Roleplaying
- Prototyping

Advantages of Requirements Engineering Process

- Helps ensure that the software being developed meets the needs and expectations of the stakeholders
- 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.
- Provides a solid foundation for the development process, which helps to reduce the risk of failure.

Disadvantages of Requirements Engineering Process:

- Can be time-consuming and costly.
- Can be difficult to ensure that all stakeholders' needs and expectations are considered
- 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.
- There may be conflicts between stakeholders, which can be difficult to resolve.

3.5. Object Oriented Analysis:

- OOA is a software engineering approach that focuses on identifying and modeling the objects within a system and their interactions.
- It's the first step in the object-oriented design (OOD) process.

Key Concepts in OOA:

- **Objects:** Real-world entities with properties (attributes) and behaviors (methods).
- **Classes:** Blueprints for creating objects, defining their attributes and methods.
- **Inheritance:** The ability of one class to inherit properties and methods from another.
- **Polymorphism:** The ability of objects to take on many forms, allowing different objects to respond to the same message in different ways.
- **Encapsulation:** The bundling of data (attributes) and methods that operate on that data within a single unit (class).

Steps in OOA:

1. **Identify Objects:**
   - Analyze the problem domain and identify the key objects.
   - Consider both tangible and intangible objects.
2. **Identify Attributes:**
   - Determine the properties or characteristics of each object.
   - These attributes define the state of an object.
3. **Identify Methods:**
   - Define the behaviors or actions that each object can perform.
   - These methods define the object's functionality.
4. **Identify Relationships:**
   - Determine the relationships between objects, such as inheritance, association, and aggregation.
5. **Create a Class Diagram:**
   - Visualize the classes, their attributes, methods, and relationships using a class diagram.

Benefits of OOA:

- **Modularity:** Breaks down complex systems into smaller, manageable units.
- **Reusability:** Promotes code reuse by creating reusable classes.
- **Maintainability:** Makes software easier to understand, modify, and extend.
- **Flexibility:** Adapts to changing requirements more easily.

Challenges of OOA:

- **Complexity:** Can be challenging to model complex systems with many interacting objects.
- **Learning Curve:** Requires understanding of object-oriented concepts.
- **Design Decisions:** Making appropriate design decisions can be difficult.

Tools for OOA:

- **UML (Unified Modeling Language):** A standard language for modeling software systems.
- **Rational Rose:** A popular CASE tool for OOA and OOD.
- **Visual Paradigm:** Another powerful CASE tool for modeling.

By effectively applying OOA principles, you can create well-structured, maintainable, and efficient software systems.

For example: Lets say you're building a game:

- OOA helps you figure out all the things you need to know about the game world – the characters, their features, and how they interact.
- It's like making a map of everything important.
- OOA also helps you understand what your game characters will do. It's like writing down a script for each character.
- Every program has specific tasks or jobs it needs to do. OOA helps you list and describe these jobs.
- In our game, it could be tasks like moving characters or keeping score. It's like making a to-do list for your software.
- OOA is smart about breaking things into different parts. It splits the job into three categories: things your game knows, things your game does, and how things in your game behave.

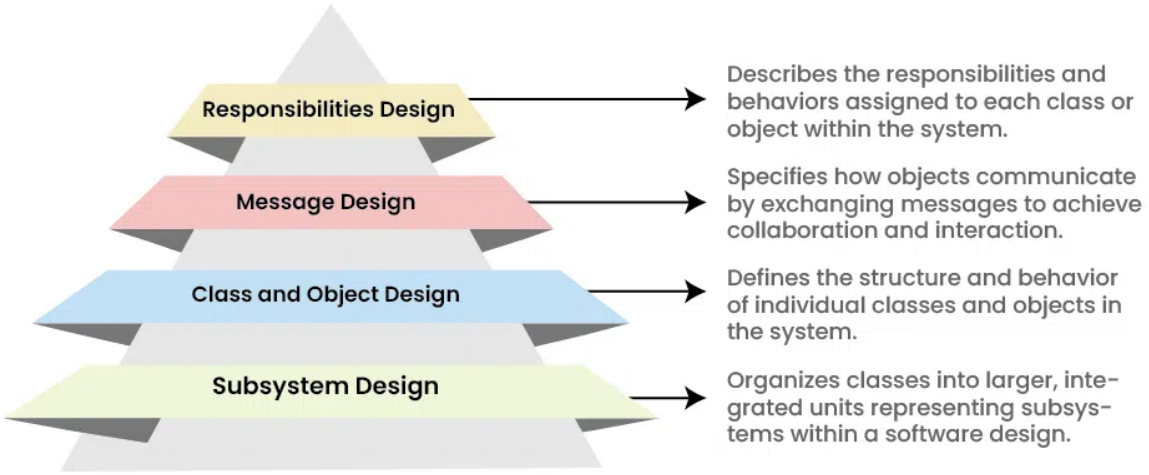
3.6. Object-Oriented Design:

- In the object-oriented software development process, the analysis model, which is initially formed through object-oriented analysis (OOA), undergoes a transformation during object-oriented design (OOD) i.e implementation of the conceptual model developed in OOA.
- This evolution is crucial because it shapes the analysis model into a detailed design model.

Furthermore, as part of the object-oriented design process, it is essential to define specific aspects:

- **Data Organization of Attributes:**
  - OOD involves specifying how data attributes are organized within the objects.
  - This includes determining the types of data each object will hold and how they relate to one another.
- **Procedural Description of Operations:**
  - OOD requires a procedural description for each operation that an object can perform.
  - This involves detailing the steps or processes involved in carrying out specific tasks.

Below diagram shows a design pyramid for object-oriented systems. It is having the following four layers.



The Object Oriented Design Pyramid

1. **The Subsystem Layer:** It represents the subsystem that enables software to achieve user requirements and implement technical frameworks that meet user needs.
2. **The Class and Object Layer:** It represents the class hierarchies that enable the system to develop using generalization and specialization. This layer also represents each object.
3. **The Message Layer:** This layer deals with how objects interact with each other. It includes messages sent between objects, method calls, and the flow of control within the system.
4. **The Responsibilities Layer:** It focuses on the responsibilities of individual objects. This includes defining the behavior of each class, specifying what each object is responsible for, and how it responds to messages.

Benefits of Object-Oriented Analysis and Design(OOAD):

- It increases the modularity and maintainability of software by encouraging the creation of tiny, reusable parts that can be combined to create more complex systems.
- It provides a high-level, abstract representation of a software system, making understanding and maintenance easier.
- It promotes object-oriented design principles and the reuse of objects, which lowers the amount of code that must be produced and raises the quality of the program.
- Software engineers can use the same language and method that OOAD provides to communicate and work together more successfully in groups.
- It can assist developers in creating scalable software systems that can adapt to changing user needs and business demands over time.

Challenges of Object-Oriented Analysis and Design(OOAD):

- Because objects and their interactions need to be carefully explained and handled, it might complicate a software system.
- Because objects must be instantiated, managed, and interacted with, this may result in additional overhead and reduce the software's speed.
- For beginner software engineers, OOAD might have a challenging learning curve since it requires a solid grasp of OOP principles and methods.
- It can be a time-consuming process that involves significant upfront planning and documentation. This can lead to longer development times and higher costs.
- OOAD can be more expensive than other software engineering methodologies due to the upfront planning and documentation required.

Real world applications of Object-Oriented Analysis and Design(OOAD):

Some examples of OOAD's practical uses are listed below:

- **Banking Software:** In banking systems, OOAD is frequently used to simulate complex financial transactions, structures, and customer interactions. Designing adaptable and reliable financial apps is made easier by OOAD's modular and scalable architecture.
- **Electronic Health Record (EHR) Systems:** Patient data, medical records, and healthcare workflows are all modeled using OOAD. Modular and flexible healthcare apps that may change to meet emerging requirements can be made through object-oriented principles.
- **Flight Control Systems:** OOAD is crucial in designing flight control systems for aircraft. It helps model the interactions between different components such as navigation systems, sensors, and control surfaces, ensuring safety and reliability.
- **Telecom Billing Systems:** In the telecom sector, OOAD is used to model and build billing systems. It enables the modular and scalable modeling of complex subscription plans, invoicing rules, and client data.
- **Online Shopping Platforms:** E-commerce system development frequently makes use of OOAD. Product catalogs, user profiles, shopping carts, and payment procedures are all modeled, which facilitates platform maintenance and functionality expansion.