Unit-6
Requirement Engineering and Modeling

What is Requirements Engineering?

A requirement is a service, function or feature that a user needs. Requirements can be functions, constraints, business rules or other elements that must be present to meet the need of the intended users.

Importance

Requirement engineering also helps in defining the scope of the software i.e. what will be the functionalities of the final software. It also helps in perceiving the cost of the final software. It also helps in perceiving the schedule up to which the software will be delivered to thecustomer

Types of Requirements:

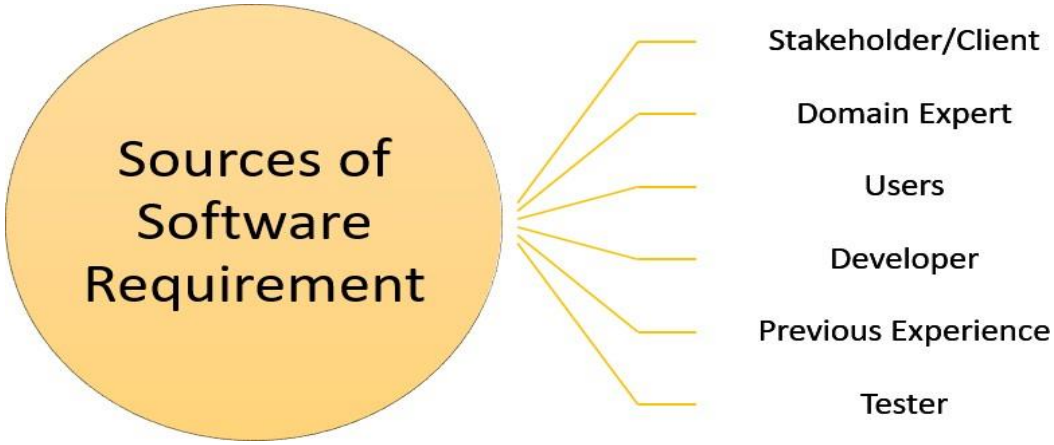
Requirements can be basically categorized into:

1. **Functional Requirements (FRs)**
   - Functional requirements express function or feature and define what is required.

2. **Non-functional Requirements (NFRs)**

Non-functional Requirements define how well, or to what level a solution needs to behave. They describe solution attributes such as security, reliability, maintainability, availability, performance and response time.

Sources of Requirements

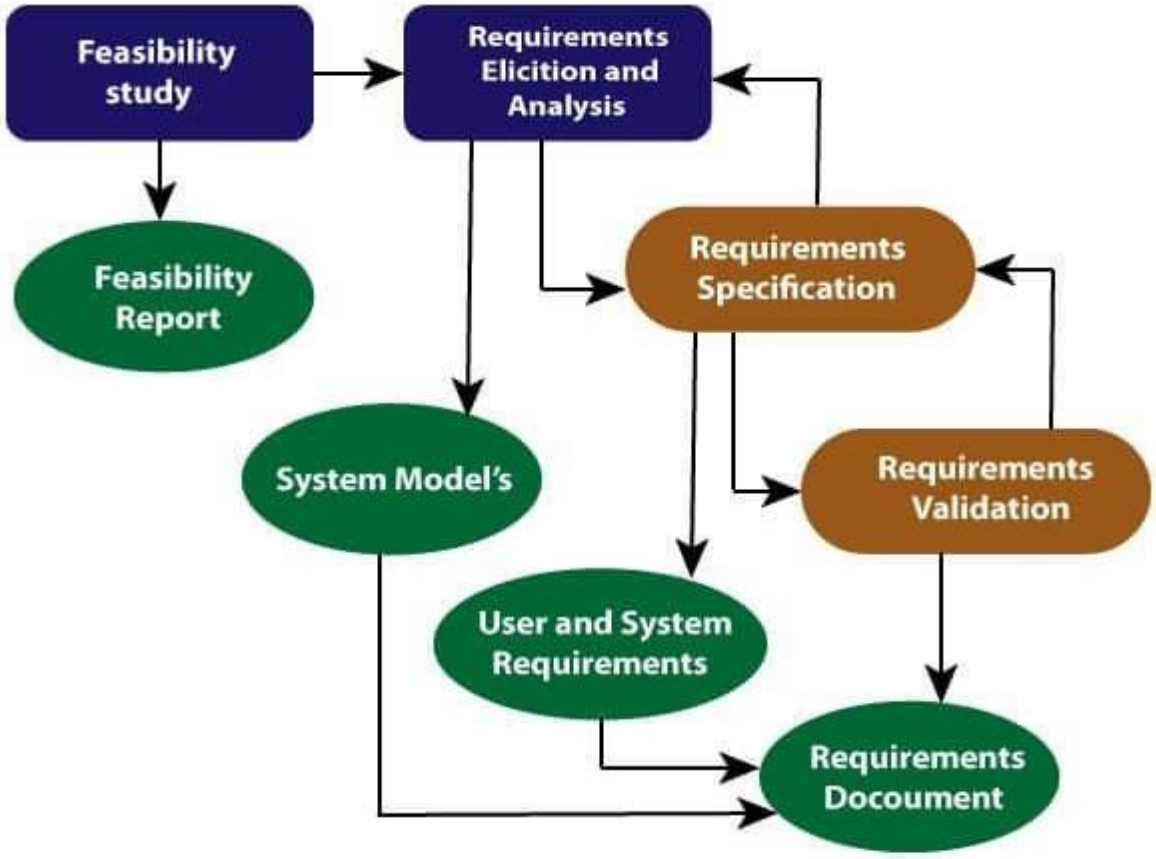


Functional Requirements	Non Functional Requirements
A functional requirement defines a system or its component.	A non-functional requirement defines the quality attribute of a software system.
It specifies "What should the software system do?"	It places constraints on "How should the software system fulfill the functional requirements?"
Functional requirement is specified by User.	Non-functional requirement is specified by technical peoples e.g. Architect, Technical leaders and software developers.
It is mandatory.	It is not mandatory.
It is captured in use case.	It is captured as a quality attribute.
Defined at a component level.	Applied to a system as a whole.
Helps you verify the functionality of the software.	Helps you to verify the performance of the software.
Functional Testing like System, Integration, End to End, API testing, etc are done.	Non-Functional Testing like Performance, Stress, Usability, Security testing, etc are done.
Usually easy to define.	Usually more difficult to define.
Example **1)** Authentication of user whenever he/she logs into the system. **2)** System shutdown in case of a cyber attack. **3)** A Verification email is sent to user whenever he/she registers for the first time on some software system.	**Example** **1)** Emails should be sent with a latency of no greater than 12 hours from such an activity. **2)** The processing of each request should be done within 10 seconds **3)** The site should load in 3 seconds when the number of simultaneous users are > 10000

Requirements Engineering Process

It 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

1. Feasibility Study:

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

Types of Feasibility:

1. Technical Feasibility - Technical feasibility evaluates the current technologies, which are needed to accomplish customer requirements within the time and budget.

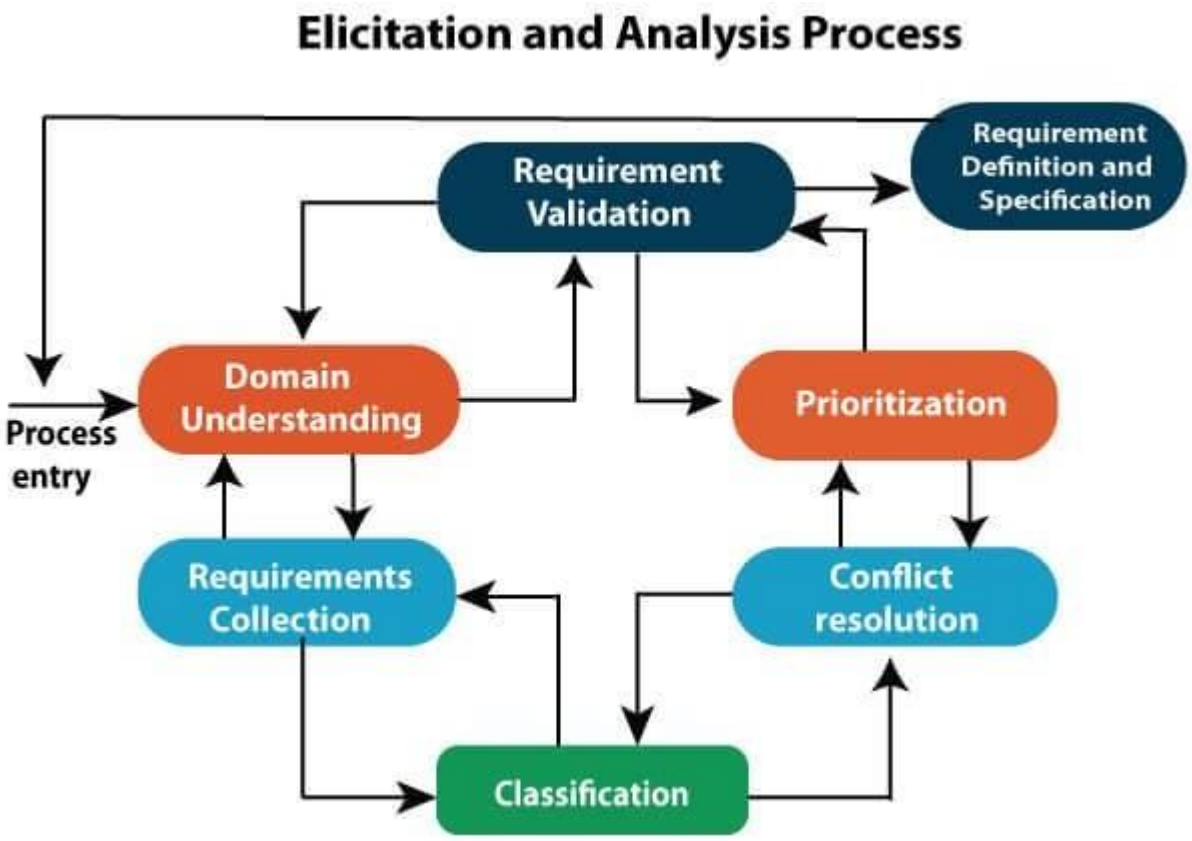
2. Operational Feasibility - Operational feasibility assesses the range in which the required software performs a series of levels to solve business problems and customer requirements.

3. Economic Feasibility - Economic feasibility decides whether the necessary software can generate financial profits for an organization.

2. Requirement Elicitation and Analysis:

This is also known as the **gathering of requirements**. Here, requirements are identified with the help of customers and existing systems processes, if available.

Analysis of requirements starts with requirement elicitation. The requirements are analyzed to identify inconsistencies, defects, omission, etc. We describe requirements in terms of relationships and also resolve conflicts if any.



3. Software Requirement Specification:

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

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

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

4. Software Requirement Validation:

After requirement specifications developed, the requirements discussed in this document are validated. The user might demand illegal, impossible solution or experts may misinterpret the needs. Requirements can be the check against the following conditions -

- If they can practically implement
- If they are correct and as per the functionality and specially of software
- If there are any ambiguities
- If they are full
- If they can describe

Requirements Validation Techniques

- **Requirements reviews/inspections:** systematic manual analysis of the requirements.
- **Prototyping:** Using an executable model of the system to check requirements.

- **Test-case generation:** Developing tests for requirements to check testability.
- **Automated consistency analysis:** checking for the consistency of structured requirements descriptions.

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.

TYPICAL REQUIREMENT ENGINEERING PROBLEMS

Understanding large and complex system requirements is difficult –
The word 'large' represents 2 aspects:

- (i) Large constraints in terms of security, etc. due to a large number of users.
- (ii) a Large number of functions to be implemented.

Undefined system boundaries –
There might be no defined set of implementation requirements. The customer may go on to include several unrelated and unnecessary functions besides the important ones, resulting in an extremely large implementation cost that may exceed the decided budget.

Customers/Stakeholders are not clear about their needs. –
Sometimes, the customers themselves may be unsure about the exhaustive list of functionalities they wish to see in the software. This might happen when they have a very basic idea about their needs but haven't planned much about the implementation part.

Changing requirements is another issue –
In the case of successive interviews or reviews from the customer, there is a possibility that the customer expresses a change in the initial set of specified requirements. While it is easy to accommodate some of the requirements, it is often difficult to deal with such changing requirements.

Partitioning the system suitably to reduce complexity –
The projects can sometimes be broken down into small modules or functionalities which are then handled by separate teams. Often, more complex and large projects require more partitioning. It needs to be ensured that the partitions are non-overlapping and independent of each other.

Validating and Tracing requirements –
Cross-checking the listed requirements before starting the implementation part is very important. Also, there should be forward as well as backward traceability. For eg, all the entity names should be the same everywhere, i.e., there shouldn't be a case where 'STUDENT' and 'STUDENTS' are used at separate places to refer to the same entity.

- **Identifying critical requirements –**

Identifying the set of requirements that have to be implemented at any cost is very important. The requirements should be prioritized so that crucial ones can be implemented first with the highest priority.

- **Proper documentation, proper meeting time, and budget constraints –**

Ensuring proper documentation is an inherent challenge, especially in the case of changing requirements. The time and budget constraints too need to be handled carefully and systematically.

UML - Overview

UML is a standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems.

UML was created by the Object Management Group (OMG) and UML 1.0 specification draft was proposed to the OMG in January 1997.

OMG is continuously making efforts to create a truly industry standard.

- UML stands for **Unified Modeling Language**.

- UML is different from the other common programming languages such as C++, Java, COBOL, etc.
- UML is a pictorial language used to make software blueprints.
- UML can be described as a general purpose visual modeling language to visualize, specify, construct, and document software system.
- Although UML is generally used to model software systems, it is not limited within this boundary. It is also used to model non-software systems as well. For example, the process flow in a manufacturing unit, etc.

UML is not a programming language but tools can be used to generate code in various languages using UML diagrams. UML has a direct relation with object oriented analysis and design. After some standardization, UML has become an OMG standard.

Types of diagram

There are two broad categories of UML diagrams and they are again divided into subcategories −

- Structural Diagrams
- Behavioral Diagrams

Structural Diagrams

The structural diagrams represent the static aspect of the system. These static aspects represent those parts of a diagram, which forms the main structure and are therefore stable.

These static parts are represented by classes, interfaces, objects, components, and nodes. The four structural diagrams are −

- Class diagram
- Object diagram
- Component diagram

- Deployment diagram

Behavioral Diagrams

Any system can have two aspects, static and dynamic. So, a model is considered as complete when both the aspects are fully covered.

Behavioral diagrams basically capture the dynamic aspect of a system. Dynamic aspect can be further described as the changing/moving parts of a system.

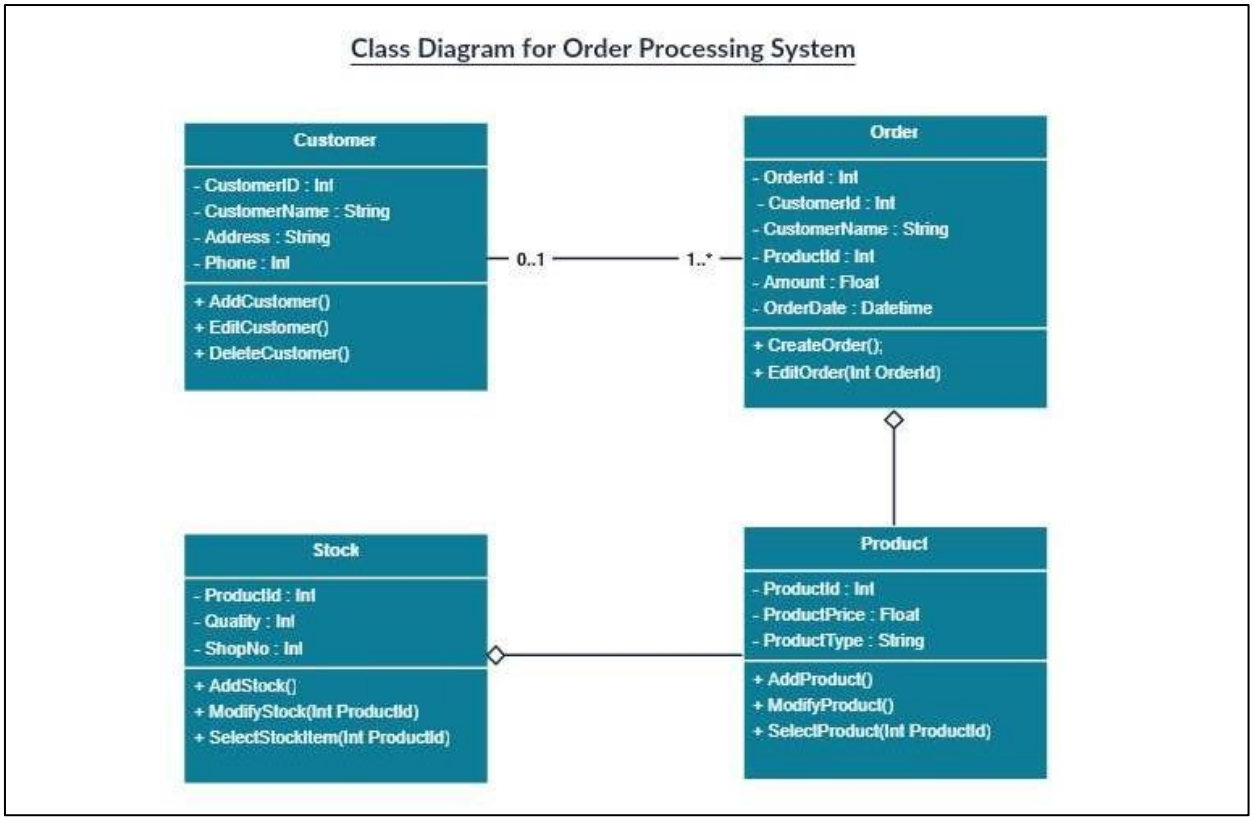
UML has the following five types of behavioral diagrams −

- Use case diagram
- Sequence diagram
- Collaboration diagram
- State chart diagram
- Activity diagram

Class Diagram

Class diagrams are the main building block of any object-oriented solution. It shows the classes in a system, attributes, and operations of each class and the relationship between each class. In most modeling tools, a class has three parts. Name at the top, attributes in the middle and operations or methods at the bottom. In a large system with many related classes, classes are grouped together to create class diagrams. Different relationships between classes are shown by different types of arrows.

Below is an image of a class diagram.

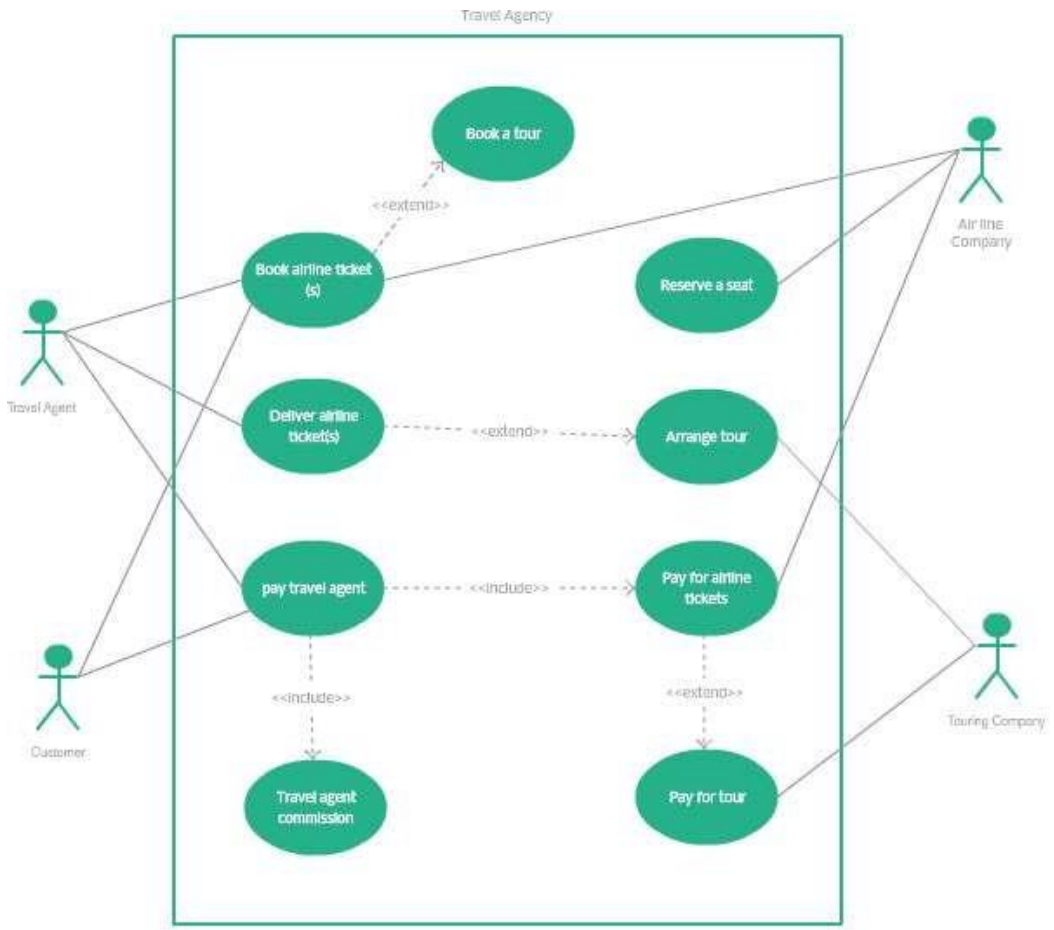


Use Case Diagram

As the most known diagram type of the behavioral UML types, Use case diagrams give a graphic overview of the actors involved in a system, different functions needed by those actors and how these different functions interact.

It's a great starting point for any project discussion because you can easily identify the main actors involved and the main processes of the system. You can create use case diagrams using our tool and/or get started instantly using our use case templates.

Use Case Diagram Relationships Explained with examples



Requirement modeling strategies

Following are the requirement modeling strategies:
1. Flow Oriented Modeling
2. Class-based Modeling

1. Flow Oriented Modeling

It shows how data objects are transformed by processing the function.

The Flow oriented elements are:

i. Data flow model

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

ii. Control flow model

- Large class applications require a control flow modeling.
- The application creates control information instated of reports or displays.
- The applications process the information in specified time.

- An event is implemented as a boolean value.

  For example, the boolean values are true or false, on or off, 1 or 0.

iii. Control Specification

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

iv. Process Specification

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

2. Class-based Modeling

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

Classes

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

Classes are found in following forms:

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

- **Structures:** The sensors, computers are defined a class of objects or related classes of objects.

Attributes

Attributes are the set of data objects that are defining a complete class within the context of the problem.

For example, 'employee' is a class and it consists of name, Id, department, designation and salary of the employee are the attributes.

Operations

The operations define the behaviour of an object.

The operations are characterized into following types:

- The operations manipulate the data like adding, modifying, deleting and displaying etc.
- The operations perform a computation.
- The operation monitors the objects for the occurrence of controlling an event.

CRS Modeling

- The CRS stands for Class-Responsibility-Collaborator.
- It provides a simple method for identifying and organizing the classes that are applicable to the system or product requirement.
- Class is an object-oriented class name. It consists of information about sub classes and super class.
- Responsibilities are the attributes and operations that are related to the class.
- Collaborations are identified and determined when a class can achieve each responsibility of it. If the class cannot identify itself, then it needs to interact with another class.