The elements are like components which can be associated in different ways to make a complete UML pictures which is known as diagram. So it is very important to understand the different diagrams to implement the knowledge in real life systems.

Any complex system is best understood by making some kind of diagrams or pictures. These diagrams have a better impact on our understanding. So if we look around then we will realize that the diagrams are not a new concept but it is used widely in different form in different industries.

We prepare UML diagrams to understand a system in better and simple way. A single diagram is not enough to cover all aspects of the system. So UML defines various kinds of diagrams to cover most of the aspects of a system.

You can also create your own set of diagrams to meet your requirements. Diagrams are generally made in an incremental and iterative way.

Structural Diagrams:

* Class diagram
* Object diagram
* Component diagram

## Class Diagram:

Class diagrams are the most common diagrams used in UML. Class diagram consists of classes, interfaces, associations and collaboration.

Class diagrams basically represent the object oriented view of a system which is static in nature.

Active class is used in a class diagram to represent the concurrency of the system.

Class diagram represents the object orientation of a system. So it is generally used for development purpose. This is the most widely used diagram at the time of system construction.

Behavioral Diagrams:

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

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
* Statechart diagram
* Activity diagram

## Use case Diagram:

Use case diagrams are a set of use cases, actors and their relationships. They represent the use case view of a system.

A use case represents a particular functionality of a system.

So use case diagram is used to describe the relationships among the functionalities and their internal/external controllers. These controllers are known as actors.

## Sequence Diagram:

A sequence diagram is an interaction diagram. From the name it is clear that the diagram deals with some sequences, which are the sequence of messages flowing from one object to another.

Interaction among the components of a system is very important from implementation and execution perspective.

So Sequence diagram is used to visualize the sequence of calls in a system to perform a specific functionality.

#### Association

An association can be named, and the ends of an association can be adorned with role names, ownership indicators, multiplicity, visibility, and other properties.

There are four different types of association: bi-directional, uni-directional, Aggregation (includes Composition aggregation) and Reflexive. Bi-directional and uni-directional associations are the most common ones.

##### Aggregation

[*Aggregation*](http://en.wikipedia.org/wiki/Aggregation_(object-oriented_programming)) is a variant of the "has a" or association relationship; aggregation is more specific than association.

An aggregation may not involve more than two classes.

*Aggregation* can occur when a class is a collection or container of other classes, but where the contained classes do not have a strong *life cycle dependency* on the container—essentially, if the container is destroyed, its contents are not.

##### Composition

[*Composition*](http://en.wikipedia.org/wiki/Object_composition) is a stronger variant of the "owns a" or association relationship; composition is more specific than aggregation.

*Composition* usually has a strong *life cycle dependency* between instances of the container class and instances of the contained class(es): If the container is destroyed, normally every instance that it contains is destroyed as well.

It always implies a multiplicity of 1 or 0..1, as no more than one object at a time can have lifetime responsibility for another object.

#### Generalization

The [*superclass*](http://en.wikipedia.org/wiki/Superclass)(base class) in the generalization relationship is also known as the *"parent"*, *superclass*, *base class*, or *base type*.

The [*subtype*](http://en.wikipedia.org/wiki/Subtype) in the specialization relationship is also known as the *"child"*, *subclass*, *derived class*, *derived type*, *inheriting class*, or *inheriting type*.