Unit-1
Introduction to UML

Importance of modeling:

"A model is a simplification of reality."

A successful software organization is one that consistently deploys quality software that meets the needs of its users. An organization that can develop such software in a timely and predictable fashion, with an efficient and effective use of resources, both human and material, is one that has a sustainable business.

Unsuccessful software projects fail in their own unique ways, but all successful projects are alike in many ways. There are many elements that contribute to a successful software organization; one common thread is the use of modeling.

"Modeling is a proven and well-accepted engineering technique."

We build architectural models of houses and high rises to help their users visualize the final product. We may even build mathematical models in order to analyze the effects of winds or earthquakes on our buildings.

Why do we model? There is one fundamental reason.

"We build models so that we can better understand the system we are developing."

Through modeling, we achieve four aims:

1. Models help us to visualize a system as it is or as we want it to be.
2. Models permit us to specify the structure or behavior of a system.
3. Models give us a template that guides us in constructing a system.
4. Models document the decisions we have made.

Why do we model the large and complex systems?

"We build models of complex systems because we cannot comprehend such a system in its entirety."

Principles of modeling:

The use of modeling has a rich history in all the engineering disciplines. That experience suggests four basic principles of modeling.

1. The choice of what models to create has a profound influence on how a Problem is attacked and how a solution is shaped.
2. Every model may be expressed at different levels of precision.
3. The best models are connected to reality.
4. No single model is sufficient. Every nontrivial system is best approached through a small set of nearly independent models.

Object oriented modeling:

In software, there are several ways to approach a model. The two most common ways are from an algorithmic perspective and from an object-oriented perspective.

Algorithmic perspective:

Algorithm perspective is the traditional view of software development. In this approach, the main building block of all software is the procedure or function.

This view leads developers to focus on issues of control and the decomposition of larger algorithms into smaller ones.

As requirements change and the system grows, systems built with an algorithmic focus turn out to be very hard to maintain.

- Language dependent.

- Algorithm/Process dependent.

Object-oriented perspective:

Object-oriented perspective is the contemporary view of software development. In this approach, the main building block of all software systems is the object or class.

An object is a thing, generally drawn from the vocabulary of the problem space or the solution space; a class is a description of a set of common objects.

Every object has identity (name, distinguish it from other objects), state (some data associated with it), and behavior (do things to other objects, as well).

With Object-oriented perspective (diagrammatic representation) it is very easy to visualize the system.

- No language dependent.
- No process dependent.

Conceptual model of the UML:

To understand conceptual model of UML first we need to clarify what is a conceptual model? And why a conceptual model is at all required?

- A conceptual model can be defined as a model which is made of concepts and their relationships.
- A conceptual model is the first step before drawing a UML diagram. It helps to understand the entities in the real world and how they interact with each other.

As UML describes the real time systems it is very important to make a conceptual model and then proceed gradually. Conceptual model of UML can be mastered by learning the following three major elements:

- UML building blocks.
- Rules to connect the building blocks.
- Common mechanisms of UML.

Building blocks of the UML

The vocabulary of the UML encompasses three kinds of building:

1. Things.
2. Relationships.
3. Diagrams.

Things:

Things are the abstractions that are first-class citizens in a model; relationships tie these things together; diagrams group interesting collections of things.

Things are the most important building blocks of UML. Things can be:

1. Structural things.
2. Behavioral things.
3. Grouping things.
4. Annotational things.

These things are the basic object-oriented building blocks of the UML. You use them to write well-formed models.

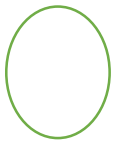
Structural Things: Structural things are the nouns of UML models. These are the mostly static parts of a model, representing elements that are either conceptual or physical. In all, there are seven kinds of structural things.

Class: a class is a description of a set of objects that share the same attributes, operations, relationships, and semantics. A class implements one or more interfaces. Graphically, a class is rendered as a rectangle.

Class
Attributes
Operations

Interface: Interface defines a set of operations which specify the responsibility of a class or component.

An interface therefore describes the externally visible behavior of that element. An interface might represent the complete behavior of a class or component or only a part of that behavior.



Ispelling

Collaboration: Collaboration defines an interaction and is a society of roles and other elements that work together to provide some cooperative behavior that's bigger than the sum of all the elements. Therefore, collaborations have structural, as well as behavioral, dimensions. A given class might participate in several collaborations.

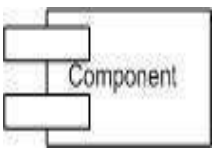
Chain of Responsibility

Use case: Use case represents a set of actions performed by a system for a specific goal. A use case is used to structure the behavioral things in a model.

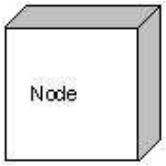
Active Class: an active class is a class whose objects own one or more processes or threads and therefore can initiate control activity. An active class is just like a class except that its objects represent elements whose behavior is concurrent with other elements.

Event Manager
Suspend() Flush()

Component: A component is a physical and replaceable part of a system that conforms to and provides the realization of a set of interfaces.



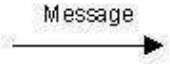
Node: N*ode* is a physical element that exists at run time and represents a



computational resource, generally having at least some memory and, often, processing capability.

Behavioral things: A behavioral thing consists of the dynamic parts of UMLmodels. Following are the behavioral things:

Interaction: Interaction is defined as a behavior that consists of a group of messages exchanged among elements to accomplish a specific task.



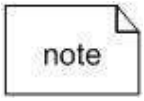
State machine: State machine is useful when the state of an object in its life cycle is important. It defines the sequence of states an object goes through in response to events. Events are external factors responsible for state change.

Grouping things: Grouping things are the organizational parts of UML models. There is only one grouping thing available:

Package: Package is the only one grouping thing available for gathering structural and behavioral things.

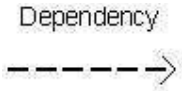
Annotational things: Annotational things are the explanatory parts of UML models. These are the comments we may apply to illuminate, describe and capture remarks of UML model elements.

Note: is the only one Annotational thing available. A note is used to render comments, constraints etc of an UML element.

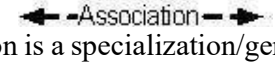


Relationship: It is another most important building block of UML. It shows how elements are associated with each other and this association describes the functionality of an application. There are four kinds of relationships available.

Dependency: Dependency is a semantic relationship between two things in which change in one element may affects the other one.

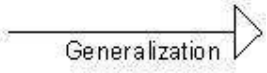


Association: Association is structural relationship that describes a set of links that connects elements of an UML model. It also describes how many objects are taking part in that relationship. We can have unidirectional association as well as bi-directional association.

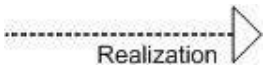


Generalization: Generalization is a specialization/generalization relationship in which objects of the specialized element (the child) are substitutable for objects of the generalized element (the parent). In this way, the child shares the structure and the behavior of the parent.

Graphically, a generalization relationship is rendered as a solid line with a hollow arrowhead pointing to the parent.



Realization: Realization is a semantic relationship between classifiers, wherein one classifier specifies a contract that another classifier guarantees to carry out. Graphically, a realization relationship is rendered as a cross between a generalization and a dependency relationship.



UML Diagrams:

A diagram is the graphical presentation of a set of elements, most often rendered as a connected graph of vertices (things) and arcs (relationships). UML diagrams are the ultimate output of the entire discussion. All the elements, relationships are used to make a complete UML diagram and the diagram represents a system.

The visual effect of the UML diagram is the most important part of the entire process. All the other elements are used to make it a complete one.

UML includes the following nine diagrams and the details are described in the following chapters.

- Class diagram
- Object diagram
- Use case diagram
- Sequence diagram
- Collaboration diagram
- Activity diagram
- Statechart diagram
- Deployment diagram
- Component diagram

Rules of the UML:

The UML's building blocks can't simply be thrown together in a random fashion. Like any language, the UML has a number of rules that specify what a well-formed model should look like. A well-formed model is one that is semantically self-consistent and in harmony with all its related models.

The UML has semantic rules for

- Name : What you can call things, relationships, and diagrams
- Scope: The context that gives specific meaning to a name
- Visibility: How those names can be seen and used by others
- Integrity: How things properly and consistently relate to one another.
- Execution: What it means to run or simulate a dynamic model.

Models built during the development of a software-intensive system tend to evolve and may be viewed by many stakeholders in different ways and at different times. For this reason, it is common for the development team to not only build models that are well-formed, but also to build models that are:

- Elided: Certain elements are hidden to simplify the view
- Incomplete: Certain elements may be missing
- Inconsistent: The integrity of the model is not guaranteed

The rules of the UML encourage youbut do not force youto address the most important analysis, design, and implementation questions that push such models to become well-formed over time.

Common Mechanisms in the UML:

A building is made simpler and more harmonious by the conformance to a pattern of common features. A house may be built in the Victorian or French country style largely by using certain architectural patterns that define those styles. The same is true of the UML. It is made simpler by the presence of four common mechanisms that apply consistently throughout the language.

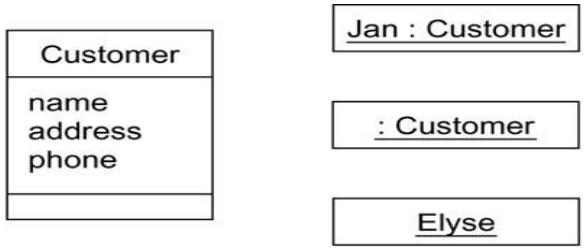
1. Specifications
2. Adornments
3. Common divisions
4. Extensibility mechanisms

Specifications: The UML is more than just a graphical language. Rather, behind every part of its graphical notation there is a specification that provides a textual statement of the syntax and semantics of that building block.

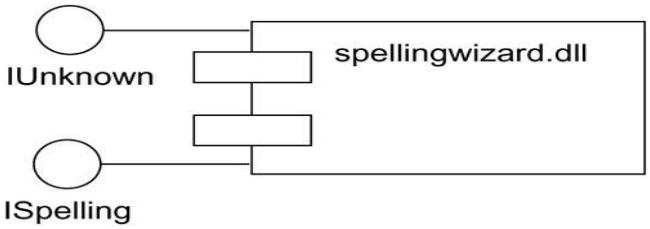
Adornments: Most elements in the UML have a unique and direct graphical notation that provides a visual representation of the most important aspects of the element.

Common Divisions: In modeling object-oriented systems, the world often gets divided in at least a couple of ways.

- There is the division of class and object. A class is an abstraction; an object is one concrete manifestation of that abstraction.



- Second, there is the separation of interface and implementation. An interface declares a contract, and an implementation represents one concrete realization of that contract, responsible for faithfully carrying out the interface's complete semantics.



Extensibility Mechanisms: The UML provides a standard language for writing software blueprints, but it is not possible for one closed language to ever be sufficient to express all possible nuances of all models across all domains across all time.The UML's extensibility mechanisms include:

Stereotypes: A stereotype extends the vocabulary of the UML, allowing you to create new kinds of building blocks that are derived from existing ones but that are specific to your problem.

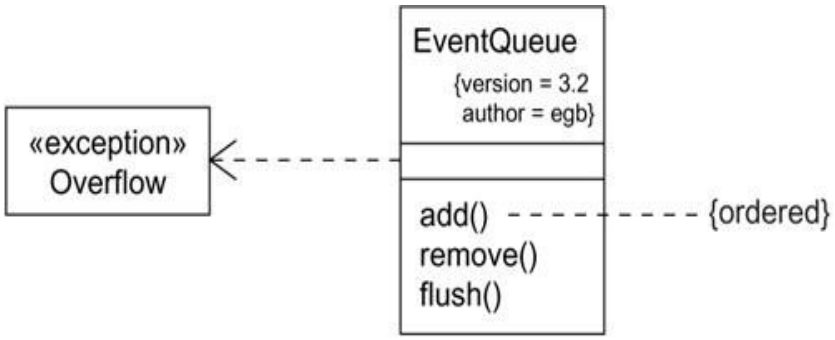
Represents with << >>

Tagged values: A tagged value extends the properties of a UML building block, allowing you to create new information in that element's specification.

Represents with { }

Constraints: A constraint extends the semantics of a UML building block, allowing you to add new rules or modify existing ones.

Represents with { }



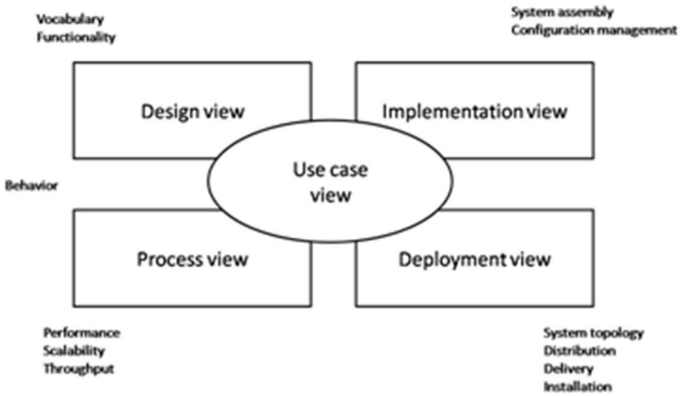
Architecture of UML:

Visualizing, specifying, constructing, and documenting a software-intensive system demands that the system be viewed from a number of perspectives.

Architecture is the set of significant decisions about

- The organization of a software system.
- The selection of the structural elements and their interfaces by which the system is composed
- Their behavior, as specified in the collaborations among those elements.
- The composition of these structural and behavioral elements into progressively larger subsystems.
- The architectural style that guides this organization: the static and dynamic elements and their interfaces, their collaborations, and their composition.

The architecture of a software-intensive system can best be described by five interlocking views. Each view is a projection into the organization and structure of the system, focused on a particular aspect of that system.



The **use case view** of a system encompasses the use cases that describe the behavior of the system as seen by its end users, analysts, and testers. This view doesn't really specify the organization of a software system.

The **design view** of a system encompasses the classes, interfaces, and collaborations that form the vocabulary of the problem and its solution. This view primarily supports the functional requirements of the system, meaning the services that the system should provide to its end users.

The **process view** of a system encompasses the threads and processes that form the system's concurrency and synchronization mechanisms. This view primarily addresses the performance, scalability, and throughput of the system.

The **implementation view** of a system encompasses the components and files that are used to assemble and release the physical system. This view primarily addresses the configuration management of the system's releases, made up of somewhat independent components and files that can be assembled in various ways to produce a running system.

The **deployment view** of a system encompasses the nodes that form the system's hardware topology on which the system executes. This view primarily addresses the distribution, delivery, and installation of the parts that make up the physical system.

Each of these five views can stand alone so that different stakeholders can focus on the issues of the system's architecture that most concern them. These five views also interact with one another nodes in the deployment view hold components in the implementation view that, in turn, represent the physical realization of classes, interfaces, collaborations, and active classes from the design and process views. The UML permits you to express every one of these five views and their interactions.