Object-Oriented Analysis

Grady Booch has defined OOA as,

“Object-oriented analysis is a method of analysis that examines requirements from the perspective of the classes and objects found in the vocabulary of the problem domain”.

Object-Oriented Analysis

The primary tasks in object-oriented analysis (OOA) are –

- Identifying objects
- Organizing the objects by creating object model diagram
- Defining the internals of the objects, or object attributes
- Defining the behavior of the objects, i.e., object actions
- Describing how the objects interact

The common models used in OOA are use cases and object models.

Object-Oriented Design

Grady Booch has defined object-oriented design as

“a method of design encompassing the process of object-oriented decomposition and a notation for depicting both logical and physical as well as static and dynamic models of the system under design”.

Object-Oriented Design

The implementation details generally include −

- Restructuring the class data (if necessary),
- Implementation of methods, i.e., internal data structures and algorithms,
- Implementation of control, and
- Implementation of associations.

Object-Oriented Programming

Grady Booch has defined object–oriented programming as

“a method of implementation in which programs are organized as cooperative collections of objects, each of which represents an instance of some class, and whose classes are all members of a hierarchy of classes united via inheritance relationships”.

Object-Oriented Programming

The important features of object–oriented programming are −

- Bottom–up approach in program design
- Programs organized around objects, grouped in classes
- Focus on data with methods to operate upon object’s data
- Interaction between objects through functions
- Reusability of design through creation of new classes by adding features to existing classes

Object-Oriented Programming

Object-oriented programming (OOP) is a programming paradigm based upon objects (having both data and methods) that aims to incorporate the advantages of modularity and reusability. Objects, which are usually instances of classes, are used to interact with one another to design applications and computer programs.

Object-Oriented Programming

Some examples of object-oriented programming languages are

- C++,
- Java,
- Smalltalk,
- Delphi,
- C#,
- Perl,
- Python,
- Ruby,
- PHP.

Importance of Modeling

A model is a simplification of reality.

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

Through modeling, we achieve four aims:

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

Principles of Modeling

Modeling has rich history in all the engineering disciplines.

That experience suggests four basic principles of modeling.

First principle of modeling

The choice of what models to create has a profound influence on how a problem is attacked and how a solution is shaped.

Choose your models well. The right models will highlight the most nasty development problems. Wrong models will mislead you, causing you to focus on irrelevant issues.

Second principle of modeling

Every model may be expressed at different levels of precision. Sometimes, a quick and simple executable model of the user interface is exactly what you need. At other times, you have to get down to complex details such as cross-system interfaces or networking issues etc.

In any case, the best kinds of models are those that let you choose your degree of detail, depending on who is viewing it. An analyst or an end user will want to focus on issues of what and a developer will want to focus on issues of how.

Third principle of modeling

The best models are connected to reality.

In software, the gap between the analysis model and the system's design model must be less. Failing to bridge this gap causes the system to diverge over time. In object-oriented systems, it is possible to connect all the nearly independent views of a system into one whole.

Fourth principle of modeling

No single model is sufficient.

Every non-trivial system is best approached through a small set of nearly independent models.

In the case of a building, you can study electrical plans in isolation, but you can also see their mapping to the floor plan and perhaps even their interaction with the routing of pipes in the plumbing plan.

What is UML?

- 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

- 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.

Goals of UML

- There are a number of goals for developing UML but the most important is to define some general purpose modeling language, which all modelers can use and it also needs to be made simple to understand and use.
- UML diagrams are not only made for developers but also for business users, common people, and anybody interested to understand the system. The system can be a software or non-software system.

- Thus it must be clear that UML is not a development method rather it accompanies with processes to make it a successful system.
- In conclusion, the goal of UML can be defined as a simple modeling mechanism to model all possible practical systems in today’s complex environment.

Conceptual Model of UML

- 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.

The 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

Role of UML in OO Design

- UML is a modeling language used to model software and non-software systems.
- Although UML is used for non-software systems, the emphasis is on modeling OO software applications.
- Most of the UML diagrams discussed so far are used to model different aspects such as static, dynamic, etc.
- Now whatever be the aspect, the artifacts are nothing but objects.
- If we look into class diagram, object diagram, collaboration diagram, interaction diagrams all would basically be designed based on the objects.

- Hence, the relation between OO design and UML is very important to understand.
- The OO design is transformed into UML diagrams according to the requirement.
- Before understanding the UML in detail, the OO concept should be learned properly.
- Once the OO analysis and design is done, the next step is very easy.
- The input from OO analysis and design is the input to UML diagrams.

UML building blocks

The building blocks of UML can be defined as −

- Things
- Relationships
- Diagrams

Things

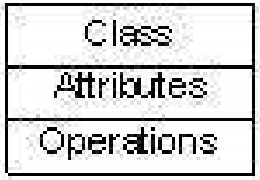
Things are the most important building blocks of UML. Things can be

- Structural
- Behavioral
- Grouping
- Annotational

Structural Things

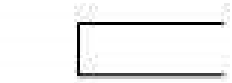
Structural things define the static part of the model. They represent the physical and conceptual elements. Following are the brief descriptions of the structural things.

Class – Class represents a set of objects having similar responsibilities.



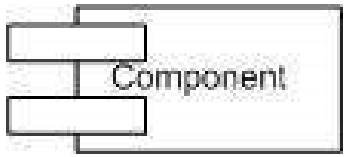
Interface – Interface defines a set of operations, which specify the responsibility of a class.

Collaboration −Collaboration defines an interaction between elements.

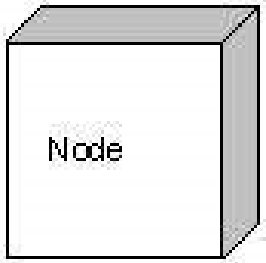


Use case −Use case represents a set of actions performed by a system for a specific goal.

Component −Component describes the physical part of a system.



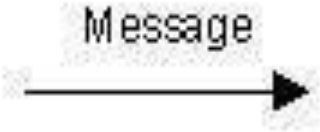
Node − A node can be defined as a physical element that exists at run time.



Behavioral Things

A behavioral thing consists of the dynamic parts of UML models. 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

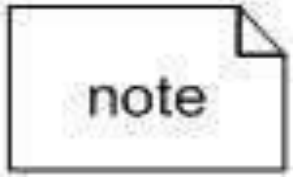
Grouping Things

Grouping things can be defined as a mechanism to group elements of a UML model together. 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 can be defined as a mechanism to capture remarks, descriptions, and comments of UML model elements. Note - It is the only one Annotational thing available.
- A note is used to render comments, constraints, etc. of an UML element.



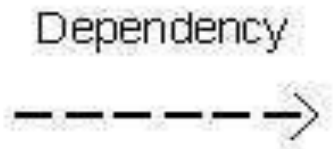
Relationship

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

There are four kinds of relationships available.

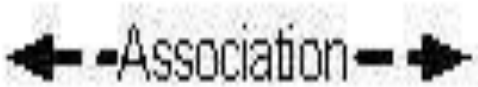
Dependency

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



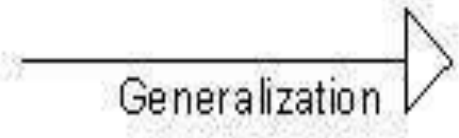
Association

Association is basically a set of links that connects the elements of a UML model. It also describes how many objects are taking part in that relationship.



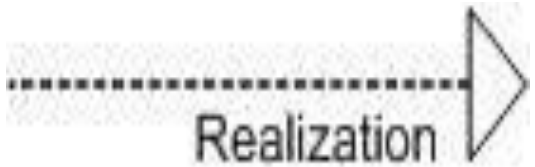
Generalization

Generalization can be defined as a relationship which connects a specialized element with a generalized element. It basically describes the inheritance relationship in the world of objects.



Realization

Realization can be defined as a relationship in which two elements are connected. One element describes some responsibility, which is not implemented and the other one implements them. This relationship exists in case of interfaces.



UML Diagrams

All the elements, relationships are used to make a complete UML diagram and the diagram represents a system

UML includes the following nine diagrams

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

UML - Architecture

- Any real-world system is used by different users.
- The users can be developers, testers, business people, analysts, and many more.
- Hence, before designing a system, the architecture is made with different perspectives in mind.
- The most important part is to visualize the system from the perspective of different viewers.
- The better we understand the better we can build the system.

UML plays an important role in defining different perspectives of a system. These perspectives are –

- Design
- Implementation
- Process
- Deployment

The center is the **Use Case** view which connects all these four. A **Use Case** represents the functionality of the system. Hence, other perspectives are connected with use case.

Design of a system consists of classes, interfaces, and collaboration. UML provides class diagram, object diagram to support this.

Implementation defines the components assembled together to make a complete physical system. UML component diagram is used to support the implementation perspective.

Process defines the flow of the system. Hence, the same elements as used in Design are also used to support this perspective.

Deployment represents the physical nodes of the system that forms the hardware. UML deployment diagram is used to support this perspective.

UML - Modeling Types

It is very important to distinguish between the UML model. Different diagrams are used for different types of UML modeling.

There are three important types of UML modeling.

Structural Modeling

Structural modeling captures the static features of a system. They consist of the following −

- Classes diagrams
- Objects diagrams
- Deployment diagrams
- Package diagrams
- Composite structure diagram
- Component diagram

- Structural model represents the framework for the system and this framework is the place where all other components exist.
- Hence, the class diagram, component diagram and deployment diagrams are part of structural modeling.
- They all represent the elements and the mechanism to assemble them.

Behavioral Modeling

Behavioral model describes the interaction in the system. It represents the interaction among the structural diagrams. Behavioral modeling shows the dynamic nature of the system.

They consist of the following

- Activity diagrams
- Interaction diagrams
- Use case diagrams

Architectural Modeling

- Architectural model represents the overall framework of the system.
- It contains both structural and behavioral elements of the system. Architectural model can be defined as the blueprint of the entire system.
- Package diagram comes under architectural modeling.

UML - Standard Diagrams

There are two broad categories of 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

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. Hence, it is generally used for development purpose.
- This is the most widely used diagram at the time of system construction.

Object Diagram

- Object diagrams can be described as an instance of class diagram. Thus, these diagrams are more close to real-life scenarios where we implement a system.
- Object diagrams are a set of objects and their relationship is just like class diagrams. They also represent the static view of the system.
- The usage of object diagrams is similar to class diagrams but they are used to build prototype of a system from a practical perspective.

Component Diagram

- Component diagrams represent a set of components and their relationships.
- These components consist of classes, interfaces, or collaborations. Component diagrams represent the implementation view of a system.
- During the design phase, software artifacts (classes, interfaces, etc.) of a system are arranged in different groups depending upon their relationship. Now, these groups are known as components.
- Finally, it can be said component diagrams are used to visualize the implementation.

Deployment Diagram

- Deployment diagrams are a set of nodes and their relationships. These nodes are physical entities where the components are deployed.
- Deployment diagrams are used for visualizing the deployment view of a system. This is generally used by the deployment team.

Behavioral Diagrams

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

Collaboration Diagram

- Collaboration diagram is another form of interaction diagram. It represents the structural organization of a system and the messages sent/received. Structural organization consists of objects and links.
- The purpose of collaboration diagram is similar to sequence diagram. However, the specific purpose of collaboration diagram is to visualize the organization of objects and their interaction.

Statechart Diagram

- Any real-time system is expected to be reacted by some kind of internal/external events. These events are responsible for state change of the system.
- Statechart diagram is used to represent the event driven state change of a system. It basically describes the state change of a class, interface, etc.
- State chart diagram is used to visualize the reaction of a system by internal/external factors.

Activity Diagram

- Activity diagram describes the flow of control in a system. It consists of activities and links. The flow can be sequential, concurrent, or branched.
- Activities are nothing but the functions of a system. Numbers of activity diagrams are prepared to capture the entire flow in a system.
- Activity diagrams are used to visualize the flow of controls in a system. This is prepared to have an idea of how the system will work when executed.