**OBJECT ORIENTED ANALYSIS & DESIGN**

**UNIT-I**

**INTRODUCTION TO UML**

**Contents:**

* 1. Importance of Modeling
	2. Principles of Modeling
	3. Object Oriented Modeling
	4. Conceptual Model of the UML
	5. Architecture
	6. Software Development Life Cycle

## What is Unified Modeling Language?

UML is a visual language for developing software blue prints (designs). A blue print or design represents the model. For example, while constructing buildings, a designer or architect develops the building blueprints. Similarly, we can also develop blue prints for a software system.

 UML is the most commonly and frequently used language for building software system blueprints. A visual language represents a language in which we can draw things to represent the knowledge. According to UML specification, there are 9 **UML diagrams**.

1. **Importance of Modeling**

To know the importance of modeling let us assume that you are going to build a dog house, a house for your family and a high rise office for a client. In the case of a dog house you need minimal resources and the satisfaction of the dog is not that important.

In the case of building a house for your family, you need to satisfy the requirements of your family members and the amount resources are non-trivial. In the case of building a high rise office, the amount of risk is very high.

Curiously, a lot of software development organizations start out wanting to build high rises but approach the problem as if they are knocking at a dog house. Sometimes you get lucky, if you have the right people at the right moment and if all the planets align properly, then you might, get your team to create a software product that satisfies its users. This happens very rarely.

Unsuccessful software projects fail in their own unique ways, but all successful software projects are alike in many ways. There are many elements that contribute to a successful software organization; one common element 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.

Modeling is not only limited to the construction industry. Modeling is applied in the fields of aeronautics, automobile, picture, sociology, economics, software development and many more. We build models so that we can validate our theories or try out new ones with minimal risk and cost.

**What, then, is a model?** Simply put,

*“A model is a simplification of reality.”*

A good model includes those elements that have broad effect and omits those minor elements that are not relevant to the given level of abstraction. A model may be structural, emphasizing the organization of the system, or it may be behavioral, emphasizing the dynamics of the system.

**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.

The larger and more complex the system becomes, the more important modeling becomes, for one very simple reason:

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

**Explanation:**

Every project can benefit from modeling. Modeling can help the development team better visualize the plan of their system and allow them to develop more rapidly by helping them build the right thing. The more complex your project, the more likely it is that you will fail or that you will build the wrong thing if you do on modeling at all.

1. **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. **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.”*

**Explanation:**

In software, the models you choose can greatly affect your world view. If you build a system through the eyes of a database developer, you will likely focus on entity-relationship models that push behavior into triggers and stored procedures. If you build a system through the eyes of a structured analyst, you will likely end up with models that are algorithmic-centric, with data flowing from process to process.

* 1. **Second principle of modeling:**

*“Every model may be expressed at different levels of precision.”*

**Explanation:**

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. Both of these stakeholders will want to visualize a system at different levels of detail at different times.

* 1. **Third principle of modeling:**

*“The best models are connected to reality.”*

**Explanation:**

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.

* 1. **Fourth principle of modeling:**

*“No single model is sufficient. Every nontrivial system is best approached through a small set of nearly independent models.”*

**Explanation:**

If you are constructing a building, there is no single set of blueprints that reveal all its details. At the very least, you'll need floor plans, elevations, electrical plans, heating plans, and plumbing plans. You can study electric plans in isolation but they map with floor plans with routing of pipes in the plumbing plan.

The same is applicable to object-oriented systems. To understand the architecture of such a system, you need several complementary and interlocking views:

1. Use case view (exposing the requirements of the system)
2. Design view (capturing the vocabulary of the problem space and the solution space),
3. Process view (modeling the distribution of the system’s processes and threads),
4. Implementation view (addressing the physical realization of the system)
5. Deployment view (focusing on system engineering issues).

Depending on the nature of the system, some models may be more important than others.

# Object Oriented Modeling

In software field, there are several ways to approach for building a model. The two most common ways are:

1. from an**algorithmic perspective (Traditional View )**
2. from an **object oriented perspective (Contemporary View)**

****

Figure 1.1: Approaches for developing models

* 1. **Traditional View**

The traditional view of software development takes an **algorithmic perspective**. In this approach, the **main building block** of all software **is the procedure or function**. This view leads the 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.

* 1. **Contemporary View**

The contemporary view of software development takes an **object oriented perspective**. In object oriented modeling, the **main building block** of all software systems **is the object or class.** Simply put, an object is a thing, generally drawn from the elements of the problem space or the solution space. A class is a description of a set of common objects. Every object has **identity** (you can name it or otherwise distinguish it from other objects), **state** (there's generally some data associated with it), and **behavior** (you can do things to the object, and it can do things to other objects, as well).

**For example**, consider simple **three-tier architecture** for a billing system, involving a **user interface**, **middleware**, and a **database**. In the **user interface**, you will find concrete objects such as buttons, menus and dialog boxes. In the **database** you will find concrete objects such as tables representing entities from the problem domain, including customers, products and orders. In the **middle layer**, you will find objects such as transactions and business rules.

The object oriented approach to software development is a part of the mainstream development simply because it has proven to be of value in building systems in all sorts of problem domains and cover all degrees of size and complexity.