CSC 2210
Object Oriented Analysis & Design

Dr. Akinul Islam Jony

Associate Professor

Department of Computer Science, FST

American International University - Bangladesh

akinul@aiub.edu

Why We Model

Importance of Modeling

>> Modeling is a central part of all the activities that lead up to the deployment of good software.

>> We build models to communicate the desired structure and behavior of our system.

>> We build models to visualize and control the system's architecture.

>> We build models to better understand the system we are building, often exposing opportunities for simplification and reuse.

>> We build models to manage risk.

Importance of Modeling

>> **Modeling** is the designing of software applications before coding. Modeling is an Essential Part of large software projects, and helpful to medium and even small projects as well.

>> **Modeling** is a proven and well-accepted engineering technique.

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

Importance of Modeling

>> There are many elements that contribute to a successful software organization; one common thread is the use of **modeling**.

>> Unsuccessful software projects fail in their own unique ways, but all successful projects are alike in many ways.

>> We build models so that we can validate our theories or try out new ones with minimal risk and cost.

>> We do model so that we can better understand the system we are developing.

>> It is appropriate for modeling systems ranging from enterprise information systems to distributed Web-based applications and even to hard real time embedded systems.

What is a Model?

>> *A **model** is a simplification of reality.*

>> A model provides the **blueprints** of a system.

>> 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.

Reasons of Modeling

>> There is one fundamental reason of **Modeling**:

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

>> Another very simple reason of **Modeling**:

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

Benefits of Model

>> 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

Limitation of Human Ability

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

- *We build models of complex systems because we cannot full meaning of such a system in its entirety.*

>> Because there are limits to the human ability to understand **complexity**.

>> Through **modeling**, we narrow the problem we are studying by focusing on only one aspect at a time. This is essentially the approach of **"divide-and-conquer"** by Edsger Dijkstra.

>> 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 no modeling at all. All interesting and useful systems have a natural tendency to become more complex over time. So, although you might think you don't need to model today, as your system evolves you will regret that decision, after it is too late.

Principles of Modeling

>> **First Principle**:

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

>> In other words, choose your models well. The right models will brilliantly illuminate the most wicked development problems, offering insight that you simply could not gain otherwise; the wrong models will mislead you, causing you to focus on irrelevant issues.

Principles of Modeling

>> **Second Principle**:

Every model may be expressed at different levels of precision.

>> That means, in software models, sometimes a quick and simple executable model of the user interface is exactly what you need; at other times you have to get down and dirty with the bits, such as when you are specifying cross-system interfaces or wrestling with networking bottlenecks. In any case, the best kinds of models are those that let you choose your degree of detail, depending on who is doing the viewing and why they need to view it. An analyst or an end user will want to focus on issues of what; 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.

Principles of Modeling

>> **Third Principle**:

The best models are connected to reality.

>> A Model connected to reality makes it more understandable and presentable. The more abstract a model is the more it needs to be closer to reality.

Principles of Modeling

>> **Fourth Principle**:

No single model or view is sufficient. Every nontrivial system is best approached through a small set of nearly independent models with multiple viewpoints.

>> In other words, we need multiple views to capture the breadth of the system. That means, having multiple models that can be built and studied separately but that are still interrelated. For example, **object-oriented software systems**.

Architecture of OO Software System

>> To understand the architecture of such a system, you need several complementary and interlocking views:

- a **use case view** (exposing the requirements of the system)
- a **design view** (capturing the vocabulary of the problem space and the solution space)
- an **interaction view** (showing the interactions among the parts of the system and between the system and the environment)
- an **implementation view** (addressing the physical realization of the system)
- a **deployment view** (focusing on system engineering issues).

>> Each of these views may have structural, as well as behavioral, aspects. Together, these views represent the blueprints of **Object-Oriented Software System**.

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.

>> 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 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*.

>> The contemporary view of software development takes an **object-oriented perspective**.

- In this approach, the main building block of all software systems is the object or class. Simply put, an object is a thing, generally drawn from the vocabulary of the problem space or the solution space.

Object-Oriented Modeling

>> For example, a simple three-tier architecture for a billing system involves

- a user interface (objects such as buttons, menus, and dialog boxes)
- business services (objects such as transactions, business rules, customers, products, and orders)
- a database (objects, such as tables)

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

>> Furthermore, most contemporary languages, operating systems, and tools are object-oriented in some fashion, giving greater cause to view the world in terms of objects.

>> Visualizing, specifying, constructing, and documenting **object-oriented systems** is exactly the purpose of the **Unified Modeling Language (UML)**.

References

>> Grady Booch, James Rumbaugh and Ivar Jacobson. *Chapter 1: Why We Model*, The Unified Modeling Language User Guide.