

# OBJECT ORIENTED DESIGN AND MODELING

**Er. Sukhpreet Kaur – Lecturer ( CSE)**

**Er. Sushil Kamboj – Lecturer (CSE)**

Shaheed Udham Singh College of Engineering and Technology, Tangori.

*Abstract : Object – oriented Software Engineering(OOSE) is an object modeling language and methodology. The approach of using object – oriented techniques for designing a system is referred to as object – oriented design. Object – oriented development approaches are best suited to projects that will implement systems using emerging object technologies to construct, manage, and assemble those objects into useful computer applications. Object oriented design is the continuation of object- oriented analysis, continuing to center the development focus around object modeling techniques.*

## 1. Introduction

In object oriented design 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, state, and behavior. 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. Object-oriented development provides the conceptual foundation for assembling systems out of components using technology such as Java Beans or COM+. Constructing object – oriented systems is exactly the purpose of the Unified Modeling Language(UML).

## 2. Design Objects

In object – oriented analysis we concentrated on identifying the objects that represented actual data within the business design. These objects are called **entity objects**. Entity objects usually correspond to items in real life and contain information, known as attributes, that describes the different instances

of the entity. They also encapsulate those behaviors that maintain its information or attributes. An entity object is said to be persistent – meaning the object typically “ lives on” after the execution of a method. Two additional types of objects will be introduced during design. New objects will be introduced to represent a means through which the user will interface with the system. These objects are called **interface objects**. It is through the interface objects that the users communicate with the system. The use case functionality that describes the user directly interacting with the system should be placed in interface objects. It translates the user’s input into information that the system can understand and use to process the business event. It also takes data pertaining to a business event and translates the data for appropriate presentation to the user. Other types of objects that are introduced are objects that hold application or business rule logic. These objects are called **control objects**. They serve as the “traffic cop” containing the application logic or business rules of the event for managing or directing the interaction between the objects. Control objects allow the scenario to be more robust and simplify the task of maintaining that process once it is implemented.

## Object Responsibility

In design we focus on identifying the behaviors a system must support and, in turn , design the methods to perform those behaviors. Along with behaviors, we determine the responsibilities an object must have. An object responsibility is the obligation that an object has to provide a service when requested, thus corroborating with other objects to satisfy the request if required.

## Object Framework

An object framework is a set of related, interacting objects that provide a well-defined set of services for accomplishing a task.

## Component

A component is a group of objects packaged together into one unit.

### 3. Object – Oriented Design Process

In performing object – oriented analysis (OOA) we have to identify objects and use cases based on ideal conditions and independent of any hardware or software solution. During object – oriented design (OOD) we have to refine those objects and use cases to reflect the actual environment of our proposed solution.

Object- oriented design includes the following activities :

- Refining the use case model to reflect the implementation environment.
- Modeling object interactions and behavior that support the use case scenario.
- Updating the object model to reflect the implementation environment.

#### **Refining the use case model to reflect the implementation environment:**

In this iteration of use case modeling, the use cases will be refined to include details of how the actor ( or user) will actually interface with the system and how the system will respond to that stimulus to process the business event. While refining use cases is often time consuming and tedious, it must be completed. These use cases will be the basis on which subsequent user manuals and test scripts are developed during system implementation. These use cases will be used by programmers to construct application programs during systems implementation.

In the following steps we will adapt each use case to the implementation environment or “ reality” and document the results. It is important that each use case be highly detailed in describing the user interaction with the system. These refined use cases can be used by the user to validate systems design and by the programmer for process and interface specifications.

**Step 1 : Transforming the “Analysis” Use Cases to “Design” Use Cases :** In this all the identified use cases are refined to reflect the physical aspects of the implementation environment for our new system. In analysis, we concentrated on the actor – the party that initiates the business event. In design,

we begin to think in terms of “how” the business event is accomplished and by whom. Thus, we are concerned with identifying the party or “system user” that is involved in processing the business event or interacting with the system. In some cases, the actor and the system user may be the same person. Descriptions of error messages, special action buttons, possible cursor movements, and other window characteristics should be included in each design use case step. The design use case step includes references to extension and abstract use cases. The **extension use cases** extend the functionality of the original use case by extracting complex or hard to understand logic into its own use case. **Abstract use cases** are those that contain steps that are used by more than one design use case.

#### **Step 2 : Updating the Use Case Diagrams and Other Documentation to Reflect Any New Use Cases :**

After all the analysis use cases have been transformed to design use cases, it is possible that new use cases or even actors have been discovered. It is very important that we keep our documentation accurate and current. Thus, in this step the use case model diagram and the actor and use case glossaries should be updated to reflect any new information introduced in step 1.

#### **Modeling object interactions and behavior that support the use case scenario:**

In this activity we have to identify and categorize the design objects required by the functionality that was specified in each use case and identify the object interactions, their responsibilities, and their behaviors.

#### **Step 1 : Identify and Classify Use Case Design Objects :**

Earlier we learned there are three categories of design objects : interface, control, and entity. In this step we examine each design use case to identify and classify the types of objects required by the logic of the use case or business scenario.

- The interface object column contains a list of objects mentioned in the use case with which the users directly interface, such as screens , windows , and printers. The only way an actor or user can interface with a system is via an interface object. Therefore, there should be at least one interface object per actor or user.
- The control object column contains a list of objects that encapsulate application logic or business rules.

- The entity object column contains a list of objects that correspond to the business domain objects whose attributes were referenced in the use case.

**Step 2 : Identify Object Attributes :** During both analysis and design, object attributes may be discovered. In efforts to transform analysis use cases into design use cases, we begin referencing the attributes in the use case text.

**Step 3 : Model High- Level Object Interactions for a Use Case :** After identifying and categorizing the design objects involved in a use case, we need to model those objects and their interactions. Such models are called **ideal object model diagrams** and are a type of a use case model in the UML. An ideal object model diagram includes symbols to represent actors, interface, control, and entity objects, and lines that represent messages or communication between the objects.

**Step 4 : Identify Object Behaviors and Responsibilities :** Once we have identified all the objects needed to support the functionality of the use case, we shift our attention to defining their specific behaviors and responsibilities. This step involves the following tasks :

- Analyze the use cases to identify required system behaviors.
- Associate behaviors and responsibilities with objects.
- Examine object model for additional behaviors.
- Verify classifications.

**Step 5 : Model Detailed Object Interactions for a Use Case:** Once we have determined the objects behaviors and responsibilities, we can create a detailed model of how the objects will interact with each other to provide the functionality specified in each design use case. The UML provides two types of diagrams to graphically depict these interactions – a sequence diagram and a collaboration diagram. Sequence diagrams show us in great detail how the objects interact with each other over time, and collaboration diagrams show us how objects collaborate in message sequence to satisfy the functionality of a use case.

**Updating the Object Model to Reflect the Implementation Environment:**

Once we have designed the objects and their required interactions, we can refine our object model to include the behaviors or implementations methods it needs to possess.

#### **4. Additional UML Design and Implementation Diagrams**

The UML offers three additional diagrams to model design and implementation aspects of the system – activity diagrams, component diagrams, and deployment diagrams. **Activity diagrams** are similar to flowcharts in that they graphically depict the sequential flow of activities of either a business process or a use case. They are different from flowcharts in that they provide a mechanism to depict activities that occur a parallel. Because of this they are very useful for modeling actions that will be performed when an operation is executing as well as the results of those actions- such as modeling the events that cause windows to be displayed or closed. Activity diagrams are flexible in that they can be used during analysis and design.

**Components diagrams** are implementation type diagrams and are used to graphically depict the physical architecture of the software of the system. They can be used to show how programming code is divided into modules and depict the dependencies between these components.

**Deployment diagrams** are also implementation type diagrams that describe the physical architecture of the hardware and software in the system. They depict the software components, processors, and devices that make up the system's architecture.

#### **References**

1. The Unified Modeling Language User Guide by Grady Booch, James Rumbaugh, Ivar Jacobson.
2. Applying Use Cases – A Practical Guide by Geri Schneider and Jason P. Winters
3. Object Solutions – Managing the Object – Oriented Project by Grady Booch.
4. Surviving Object – Oriented Projects – A Manager Guide by Alistair Cockburn
5. Object Oriented Software Engineering – A Use Case Driven Approach by Ivar Jacobson, Magnus Christerson, Patrik Jonsson, and Gunnar Overgaard.
6. Designing Object – Oriented User Interfaces by Dave Collins.