OBJECT-ORIENTED ANALYSIS AND DESIGN

Object-oriented analysis and design (OOAD) is a software engineering approach that models a system as a group of interacting objects. Each object represents some entity of interest in the system being modeled, and is characterized by its class, its state (data elements), and its behavior. Various models can be created to show the static structure, dynamic behavior, and runtime deployment of these collaborating objects. There are a number of different notations for representing these models, such as the Unified Modeling Language (UML).

Object-oriented analysis (OOA) applies object-modeling techniques to analyze the functional requirements for a system. Object-oriented design (OOD) elaborates the analysis models to produce implementation specifications. OOA focuses on what the system does, OOD on how the system does it.

Object-Oriented Systems

An object-oriented system is composed of objects. The behavior of the system results from the collaboration of those objects. Collaboration between objects involves those sending messages to each other. Sending a message differs from calling a function in that when a target object receives a message, it itself decides what function to carry out to service that message. The same message may be implemented by many different functions, the one selected depending on the state of the target object.

The implementation of "message sending" varies depending on the architecture of the system being modeled, and the location of the objects being communicated with.

Object-oriented analysis

Object-oriented analysis (OOA) looks at the problem domain, with the aim of producing a conceptual model of the information that exists in the area being analyzed. Analysis models do not consider any implementation constraints that might exist, such as concurrency, distribution, persistence, or how the system is to be built. Implementation constraints are dealt during object-oriented design (OOD). Analysis is done before the Design.

The sources for the analysis can be a written requirements statement, a formal vision document, interviews with stakeholders or other interested parties. A system may be divided into multiple domains, representing different business, technological, or other areas of interest, each of which are analyzed separately.

The result of object-oriented analysis is a description of what the system is functionally required to do, in the form of a conceptual model. That will typically be presented as a set of use cases, one or more UML class diagrams, and a number of interaction diagrams. It may also include some kind of user interface mock-up. The purpose of object oriented analysis is to develop a model that describes computer software as it works to satisfy a set of customer defined requirements.

Object-Oriented Design

Object-oriented design (OOD) transforms the conceptual model produced in object-oriented analysis to take account of the constraints imposed by the chosen architecture and any non-
functional – technological or environmental – constraints, such as transaction throughput, response time, run-time platform, development environment, or programming language.

The concepts in the analysis model are mapped onto implementation classes and interfaces. The result is a model of the solution domain, a detailed description of how the system is to be built. Object-oriented design is 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.

There are two important parts to this definition: object-oriented design
(1) Leads to an object oriented decomposition and
(2) Uses different notations to express different models of the logical (class and object structure) and physical (module and process architecture) design of a system, in addition to the static and dynamic aspects of the system.

**Object-Oriented Programming**

Is object-oriented programming (or OOP, as it is sometimes written)? We define it as follows: Object-oriented programming is 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.

There are three important parts to this definition: object-oriented programming
(1) Uses objects, not algorithms, as its fundamental logical building;
(2) Each object is an instance of some class; and
(3) Classes are related to one another via inheritance relationships.

A program may appear to be object-oriented, but if any of these elements is missing, it is not an object-oriented program. Specifically, programming without inheritance ‘is distinctly not object-oriented; we call it programming with abstract data types.

There have been basically 3 approaches in information system development area:

- Process-oriented,
- Data-oriented and
- Object-oriented approaches.

As information technology (both hardware and software) has been advancing, people have moved from the earliest process-oriented approach to data-oriented approach and now begun to adopt the latest object-oriented analysis methodology.

Unlike its two predecessors that focus either on process or data, the object-oriented approach combines data and processes (called methods) into single entities called objects. Objects usually correspond to the real things an information system deals with, such as customers, suppliers, contracts, and rental agreements. Object-oriented model is able to thoroughly represent complex relationships and to represent data and data processing with a consistent notation, which allows an easier blending of analysis and design in an evolutionary process. The goal of object-oriented
approach is to make system elements more reusable, thus improving system quality and the productivity of systems analysis and design.

**Mechanism of Object-oriented Approach**

The principals of objects, **encapsulation, inheritance, and polymorphism** are the foundation for object-oriented systems development. To understand and express the essential and interesting features of an application in the complex real world, an object-oriented model is built around objects. An object encapsulates both data and behavior, implying that analysts can use the object-oriented approach for both data modeling and process modeling.

Specific objects in a system can inherit characteristics from the global instance of an object. For example, many types of objects may have a name and a creation date. Specific objects can inherit these global characteristics from parent objects that include only global characteristics. Objects can inherit characteristics from more than one parent object. Inheritance attempts to avoid the redundant definition of similar characteristics that can be embodied at higher levels in the system.

By a concept called polymorphism, functionality that is conceptually similar among differing objects is extracted to a global level. This process limits the production of parallel functionality and streamlines the information interface. Polymorphism directs the specification writer to understand the functionality of a process and make it available to any object that requires a similar instance of functionality.

**Unified Modeling Language**

The Unified Modeling Language (UML) is an object-oriented language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling. The UML was developed by Rational Software and its partners. It is the successor to the modeling languages found in the Booch (Booch 1994), OOSE/Jacobson, OMT and other methods.

By offering a common blueprinting language, UML relieves developers of the proprietary ties that are so common in this industry. Major vendors including IBM, Microsoft, and Oracle are brought together under the UML umbrella. And because UML uses simple, intuitive notation, nonprogrammers can also understand UML models. In fact, many of the language's supporters claim that UML's simplicity is its chief benefit. If developers, customers, and implementers can all understand a UML diagram, they are more likely to agree on the intended functionality, thereby improving their chances of creating an application that truly solves a business problem.

The UML, a visual modeling language, is not intended to be a visual programming language. The UML notation is useful for graphically depicting object-oriented analysis and design models. It not only allows you to specify the requirements of a system and capture the design decisions, but it also promotes communication among key persons involved in the development effort. The emphasis in modeling should be on analysis and design, focusing on front-end
conceptual issues, rather than back-end implementation issues, which unnecessarily restrict design choices.

**Analysis Process**

In the analysis phase, a model of the real-world application is developed showing its important properties. It abstracts concepts from the application domain and describes what the intended system must do, rather than how it will be done.

Most proponents of object-oriented analysis (OOA) claim that the use of object-oriented concepts in the analysis phase (i.e., OOA) increases the understanding of problem domains, that OOA promotes a smooth transition from the analysis phase to the design phase, and that OOA provides a more natural way of organizing specifications.

**Extant object-oriented approaches can be classified into three categories:**

1. Combinative approaches use different modeling techniques in different stages of the system development process;
2. Adaptive approaches apply existing techniques (e.g., data-flow diagram and entity-relationship approach) in object-oriented ways to analyze the problem domain; and
3. Pure approaches adopt an object-oriented perspective in systems analysis and design.

**Use-case Modeling**

First adopted by Jacobson et al. (1992), use-case modeling is developed in the analysis phase of the object-oriented system development life cycle. Use-case modeling is done in the early stages of system development to help developers gain a clear understanding of the functional requirement of the system, without worrying about how those requirements will be implemented.

A use-case is a representation of a discrete set of work performed by a use (or another system) using the operational system (). A **use-case model consists of actors and use cases**. An actor is an external entity that interacts with the system and a use case represents a sequence of related actions initiated by an actor to accomplish a specific goal.

For identifying use cases, we recommend to ask the following questions:

- What are the main tasks performed by each actor?
- Will the actor read or update any information in the system?
- Will the actor have to inform the system about changes outside the system?
- Does the actor have to be informed about unexpected changes?

In UML, a use-case model is depicted in a use-case diagram that contains the use cases and actors for a system. Begin working with the UML by modeling all scenarios in the system or business with Use Case diagrams. Describe the system in terms of actors, which are external
agents that request a service from the system, and Use Cases. Each Use Case can be defined simply by a textual statement that describes the scenario, or via other definitions, such as the sequence of steps that are performed within the scenario, or the pre- and post-conditions of the scenario.

A use-case diagram for a hotel reservations system is shown in Exhibit 1. An actor is shown using a stickman symbol with its name below. There are two actors outside the box: manager and receptionist. Inside the box are 12 use cases – run management reports, cancel unconfirmed reservation, cancel reservation, check in guest, check out guest, etc.

Exhibit 1 A use-case diagram for Hotel Reservations System (Popking 2001)

Use-cases are shown as ellipses with their names inside and are performed by the actors outside the system. A use-case is always initiated by an actor. For example, Check in Guest is initiated by Receptionist.

A use-case may interact with other use-cases. Some of these relationships include extend and use and are reflected by single hollow arrow lines. For instance, Cancel Reservation use case uses information from Customer Requests Cancellation use case or Cancel Unconfirmed Reservation use case to finalize the actual cancellation of reservations.

While a use-case diagram shows all the use cases in the system, it does not describe how those use cases are carried out by the actors. The contents of a use case are normally described in plain text. While describing a use case, you should focus on its external behavior, that is, how it interacts with the actors, rather than how the use case is performed inside the system.
Also called system usage modeling, a use case modeling, at requirements analysis stage, consists of a Use Case Diagram plus a set of descriptions as well as illustrations of prototype screens.

One of the benefits of use-case or system usage modeling is its simplicity. The strength of the technique is in its non-technical simplicity, which allows users to participate in a way that is seldom possible using the abstractions of Class Modeling alone. It also helps the analyst get to grip with specific user needs before analyzing the internal mechanics of a system. Use cases also fit particularly well within an evolutionary and incremental process in that they provide a basis for early prototyping and readily identifiable units for incremental delivery. They also provide a means of traceability for functional requirements upstream in the process and for constructing test plans downstream in the process. It is no accident therefore that use cases have become such a popular technique. In summary, provides a complete list of the steps in system usage modeling as follows:

- Identify the actors.
- Identify the use cases.
- Create a Use Case Diagram.
- Describe the use cases.
- Complete the use case descriptions.