OBJECT-ORIENTED PARADIGM

http://www.tutorialspoint.com/software_architecture_design/object_oriented_paradigm.htm

Copyright © tutorialspoint.com

The object-oriented *OO* paradigm took its shape from the initial concept of a new programming approach, while the interest in design and analysis methods came much later. OO analysis and design paradigm is the logical result of the wide adoption of OO programming languages.

Development of OO

OO is developed over a period of time as −

- The first object–oriented language was **Simula** *Simulationofrealsystems* that was developed in 1960 by researchers at the Norwegian Computing Center.
- In 1970, **Alan Kay** and his research group at Xerox PARC created a personal computer named **Dynabook** and the first pure object-oriented programming language *OOPL* - Smalltalk, for programming the Dynabook.
- In the 1980s, **Grady Booch** published a paper titled Object Oriented Design that mainly presented a design for the programming language, Ada. In the ensuing editions, he extended his ideas to a complete object–oriented design method.
- In the 1990s, **Coad** incorporated behavioral ideas to object-oriented methods.

The other significant innovations were Object Modeling Techniques *OMT* by **James Rum Baugh** and Object-Oriented Software Engineering *OOSE* by **Ivar Jacobson**.

Introduction to OO Paradigm

OO paradigm is a significant methodology for the development of any software. Most of the architecture styles or patterns such as pipe and filter, data repository, and component-based can be implemented by using this paradigm.

Basic concepts and terminologies of object–oriented systems −

Object

An object is a real-world element in an object–oriented environment that may have a physical or a conceptual existence. Each object has −

- Identity that distinguishes it from other objects in the system.
- State that determines characteristic properties of an object as well as values of properties that the object holds.
- Behavior that represents externally visible activities performed by an object in terms of changes in its state.

Objects can be modeled according to the needs of the application. An object may have a physical existence, like a customer, a car, etc.; or an intangible conceptual existence, like a project, a process, etc.

Class

A class represents a collection of objects having same characteristic properties that exhibit common behavior. It gives the blueprint or the description of the objects that can be created from it. Creation of an object as a member of a class is called instantiation. Thus, an object is an **instance** of a class.

The constituents of a class are −

- A set of attributes for the objects that are to be instantiated from the class. Generally,

different objects of a class have some difference in the values of the attributes. Attributes are often referred as class data.

- A set of operations that portray the behavior of the objects of the class. Operations are also referred as functions or methods.

Example

Let us consider a simple class, Circle, that represents the geometrical figure circle in a two–dimensional space. The attributes of this class can be identified as follows −

- x–coord, to denote x–coordinate of the center
- y–coord, to denote y–coordinate of the center
- a, to denote the radius of the circle

Some of its operations can be defined as follows −

- findArea, a method to calculate area
- findCircumference, a method to calculate circumference
- scale, a method to increase or decrease the radius

Encapsulation

Encapsulation is the process of binding both attributes and methods together within a class. Through encapsulation, the internal details of a class can be hidden from outside. It permits the elements of the class to be accessed from outside only through the interface provided by the class.

Polymorphism

Polymorphism is originally a Greek word that means the ability to take multiple forms. In object-oriented paradigm, polymorphism implies using operations in different ways, depending upon the instances they are operating upon. Polymorphism allows objects with different internal structures to have a common external interface. Polymorphism is particularly effective while implementing inheritance.

Example

Let us consider two classes, Circle and Square, each with a method findArea. Though the name and purpose of the methods in the classes are same, the internal implementation, i.e., the procedure of calculating an area is different for each class. When an object of class Circle invokes its findArea method, the operation finds the area of the circle without any conflict with the findArea method of the Square class.

Relationships

In order to describe a system, both dynamic *behavioral* and static *logical* specification of a system must be provided. The dynamic specification describes the relationships among objects e.g. message passing. And static specification describe the relationships among classes, e.g. aggregation, association, and inheritance.

Message Passing

Any application requires a number of objects interacting in a harmonious manner. Objects in a system may communicate with each other by using message passing. Suppose a system has two objects − obj1 and obj2. The object obj1 sends a message to object obj2, if obj1 wants obj2 to execute one of its methods.

Composition or Aggregation

Aggregation or composition is a relationship among classes by which a class can be made up of any combination of objects of other classes. It allows objects to be placed directly within the body of other classes. Aggregation is referred as a “part–of” or “has–a” relationship, with the ability to navigate from the whole to its parts. An aggregate object is an object that is composed of one or

more other objects.

Association

Association is a group of links having common structure and common behavior. Association depicts the relationship between objects of one or more classes. A link can be defined as an instance of an association. The Degree of an association denotes the number of classes involved in a connection. The degree may be unary, binary, or ternary.

- A unary relationship connects objects of the same class.
- A binary relationship connects objects of two classes.
- A ternary relationship connects objects of three or more classes.

Inheritance

It is a mechanism that permits new classes to be created out of existing classes by extending and refining its capabilities. The existing classes are called the base classes/parent classes/super-classes, and the new classes are called the derived classes/child classes/subclasses.

The subclass can inherit or derive the attributes and methods of the super-class *es* provided that the super-class allows so. Besides, the subclass may add its own attributes and methods and may modify any of the super-class methods. Inheritance defines a “is – a” relationship.

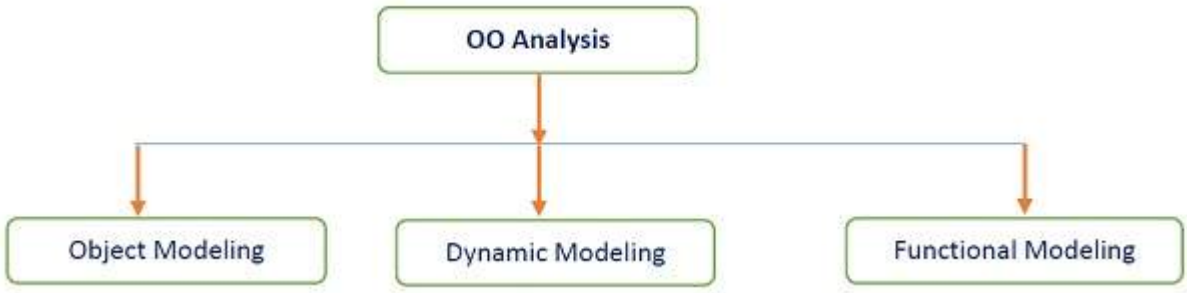
Example

From a class Mammal, a number of classes can be derived such as Human, Cat, Dog, Cow, etc. Humans, cats, dogs, and cows all have the distinct characteristics of mammals. In addition, each has its own particular characteristics. It can be said that a cow “is – a” mammal.

OO Analysis

In object-oriented analysis phase of software development, the system requirements are determined, the classes are identified, and the relationships among classes are acknowledged. The aim of OO analysis is to understand the application domain and specific requirements of the system. The result of this phase is requirement specification and initial analysis of logical structure and feasibility of a system.

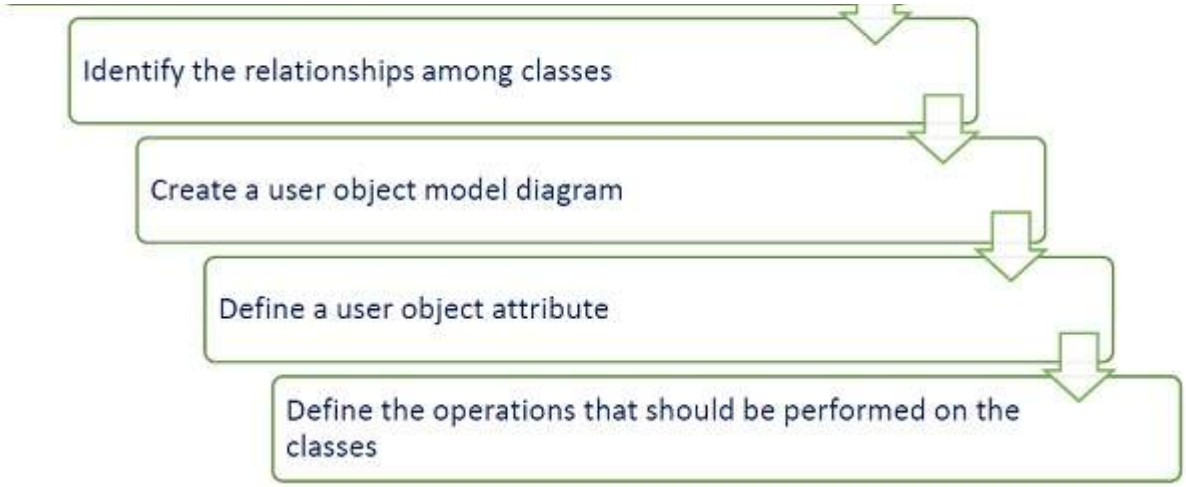
The three analysis techniques that are used in conjunction with each other for object-oriented analysis are object modeling, dynamic modeling, and functional modeling.



Object Modeling

Object modeling develops the static structure of the software system in terms of objects. It identifies the objects, the classes into which the objects can be grouped into and the relationships between the objects. It also identifies the main attributes and operations that characterize each class.

The process of object modeling can be visualized in the following steps −

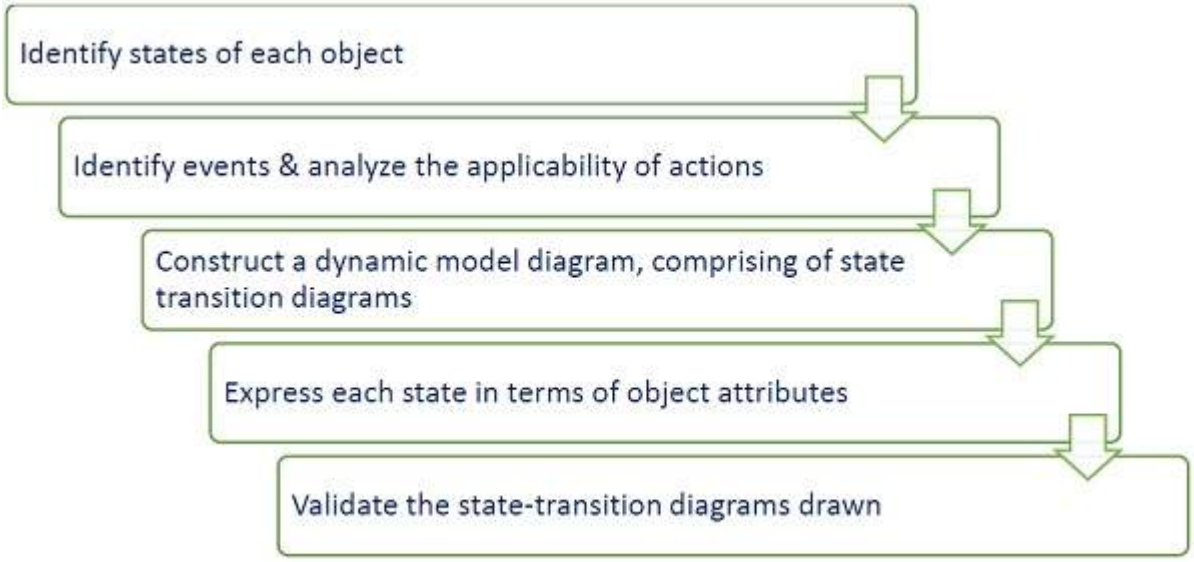


Dynamic Modeling

After the static behavior of the system is analyzed, its behavior with respect to time and external changes needs to be examined. This is the purpose of dynamic modeling.

Dynamic Modeling can be defined as “a way of describing how an individual object responds to events, either internal events triggered by other objects, or external events triggered by the outside world.”

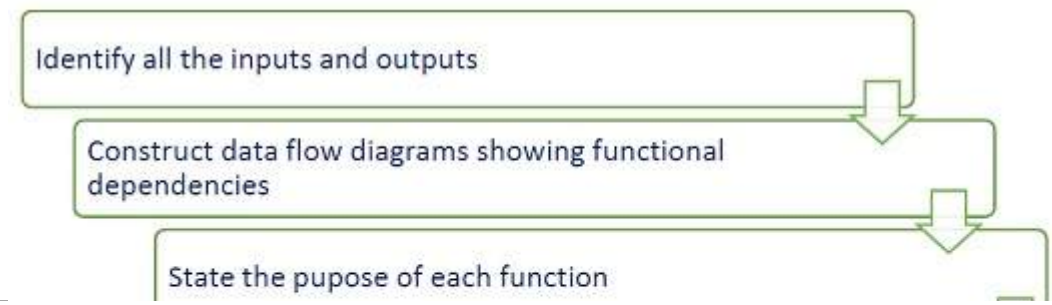
The process of dynamic modeling can be visualized in the following steps −

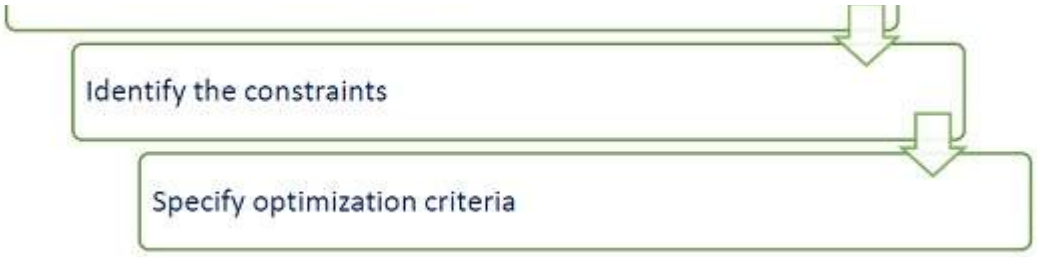


Functional Modeling

Functional Modeling is the final component of object-oriented analysis. The functional model shows the processes that are performed within an object and how the data changes, as it moves between methods. It specifies the meaning of the operations of an object modeling and the actions of a dynamic modeling. The functional model corresponds to the data flow diagram of traditional structured analysis.

The process of functional modeling can be visualized in the following steps −

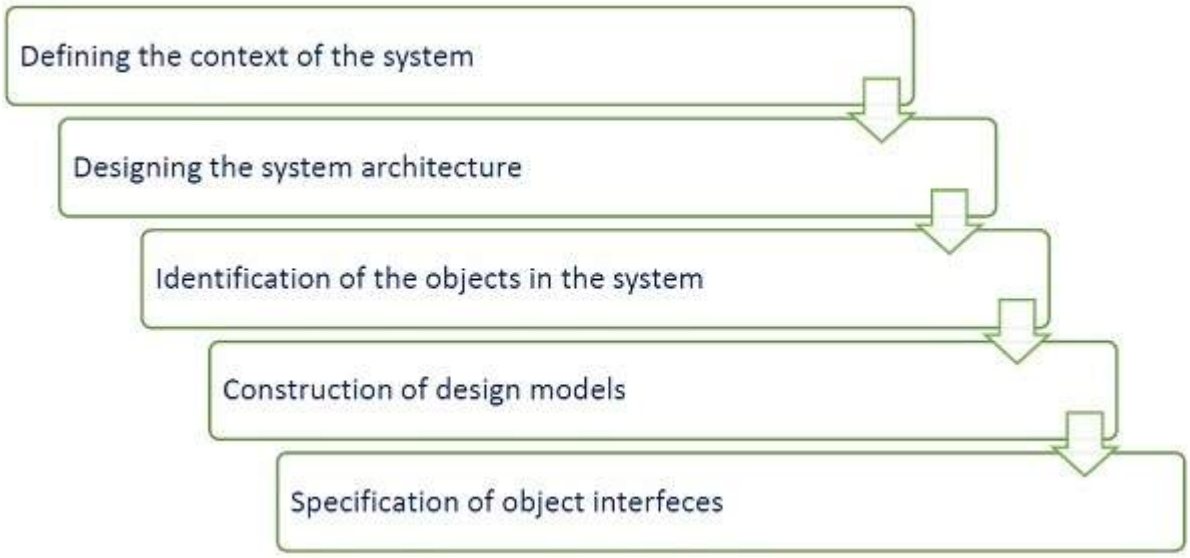




Object-Oriented Design

After the analysis phase, the conceptual model is developed further into an object-oriented model using object-oriented design *OOD*. In OOD, the technology-independent concepts in the analysis model are mapped onto implementing classes, constraints are identified, and interfaces are designed, resulting in a model for the solution domain. The main aim of OO design is to develop the structural architecture of a system.

The stages for object–oriented design can be identified as −



OO Design can be divided into two stages − Conceptual design and Detailed design.

Conceptual design

In this stage, all the classes are identified that are needed for building the system. Further, specific responsibilities are assigned to each class. Class diagram is used to clarify the relationships among classes, and interaction diagram are used to show the flow of events. It is also known as **high-level design**.

Detailed design

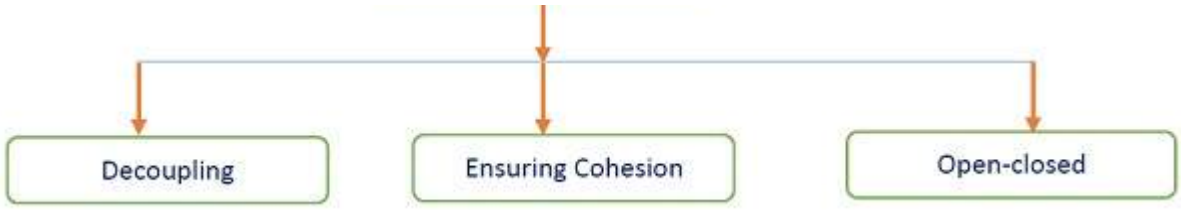
In this stage, attributes and operations are assigned to each class based on their interaction diagram. State machine diagram are developed to describe the further details of design. It is also known as **low-level design**.

Design Principles

Following are the major design principles −

Principle of Decoupling

It is difficult to maintain a system with a set of highly interdependent classes, as modification in one class may result in cascading updates of other classes. In an OO design, tight coupling can be eliminated by introducing new classes or inheritance.



Ensuring Cohesion

A cohesive class performs a set of closely related functions. A lack of cohesion means — a class performs unrelated functions, although it does not affect the operation of the whole system. It makes the entire structure of software hard to manage, expand, maintain, and change.

Open-closed Principle

According to this principle, a system should be able to extend to meet the new requirements. The existing implementation and the code of the system should not be modified as a result of a system expansion. In addition, the following guidelines have to be followed in open-closed principle −

- For each concrete class, separate interface and implementations have to be maintained.
- In a multithreaded environment, keep the attributes private.
- Minimize the use of global variables and class variables.

Loading [MathJax]/jax/output/HTML-CSS/jax.js