Comprehensive Study of Object-Oriented Analysis and Design by using the Concept of OOSE

Sanjeev Kumar Dhiman
Assistant Professor
Deptt of CSE
SRMIET, Distt. Ambala
Haryana, India

Amit Sharma
Assistant Professor
Deptt of CSE
SRMIET, Distt. Ambala
Haryana, India

Amanbir Kaur
Assistant Professor
Deptt of CSE
GIEMT,Amritsar
Punjab, India

ABSTRACT

Object Oriented Analysis is concerned with specifying system requirements and analyzing the application domain. Object Oriented Design is concerned with implementing the requirements identified during OOA in the application domain. In this paper, the brief overview of object oriented concepts, its analysis and designing concepts, their strategies with its various advantages and disadvantages have been discussed.

1. INTRODUCTION

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 run-time 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 .An object contains encapsulated data and procedures grouped together to represent an entity. The 'object interface', how the object can be interacted with, is also defined. An object-oriented program is described by the interaction of these objects. Object-oriented design is the discipline of defining the objects and their interactions to solve a problem that was identified and documented during object-oriented analysis.

2 OBJECT ORIENTED CONCEPTS

The five basic concepts of object-oriented design are the implementation level features that are built into the programming language. These features are often referred to by these common names:

(a) Object/Class: A tight coupling or association of data structures with the methods or functions that act on the data. This is called a class, or object (an object is created based on a class). Each object serves a separate function. It is defined by its properties, what it is and what it can do. An object can be part

Of a class, which is a set of objects that are similar?

(b) Information hiding: The ability to protect some components of the object from external entities. This is realized by language

keywords to enable a variable to be declared as private or protected to the owning class

(c) Inheritance: The ability for a class to extend or override functionality of another class. The so-called subclass has a whole section that is derived (inherited) from the super class and then it has its own set of functions and data.

(d) Interface: The ability to defer the implementation of a method. The ability to define the functions or methods signatures without implementing them.

(e) Polymorphism: The ability to replace an object with its sub objects. The ability of an object-variable to contain, not only that object, but also all of its sub objects.

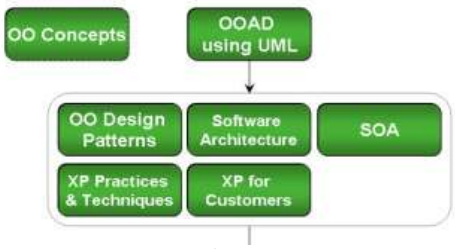

Fig 1: Object Oriented Software Engineering

3. OBJECT ORIENTED DESIGN

3.1 Input (sources) for object-oriented design

The input for object-oriented design is provided by the output of object-oriented analysis. Realize that an output artifact does not need to be completely developed to serve as input of object-oriented design; analysis and design may occur in parallel, and in practice the results of one activity can feed the other in a short feedback cycle through an iterative process. Both analysis and design can be performed incrementally, and the artifacts can be continuously grown instead of completely developed in one shot. Typical input artifacts for object-oriented design are:

3.1.1 Conceptual model: Conceptual model is the result of object-oriented analysis .it captures concepts in the problem domain. The conceptual model is explicitly chosen to be independent of implementation details, such as concurrency or data storage.

3.1.2 Use case: Use case is a description of sequences of events that, taken together, lead to a system doing something useful. Each use case provides one or more scenarios that convey how the system should interact with the users called actors to achieve a specific business goal or function. Use case actors may be end users or other systems. In many circumstances use cases are further elaborated into use case diagrams. Use case diagrams are used to identify the actor (users or other systems) and the processes they perform.

3.1.3 System Sequence Diagram: System Sequence diagram (SSD) is a picture that shows, for a particular scenario of a use case, the events that external actors generate, their order, and possible inter-system events.

3.1.4 User interface documentations (if applicable): Document that shows and describes the look and feel of the end product's user interface. It is not mandatory to have this, but it helps to visualize the end-product and therefore helps the designer.

3.1.5 Relational data model (if applicable) A data model is an abstract model that describes how data is represented and used. If an object database is not used, the relational data model should usually be created before the design, since the strategy chosen for object-relational mapping is an output of the OO design process. However, it is possible to develop the relational data model and the object-oriented design artifacts in parallel, and the growth of an artifact can stimulate the refinement of other artifacts.

3.2 Output (deliverables) of object-oriented design

3.2.1 Sequence Diagrams: Extend the System Sequence Diagram to add specific objects that handle the system events. A sequence diagram shows, as parallel vertical lines, different processes or objects that live simultaneously, and, as horizontal arrows, the messages exchanged between them, in the order in which they occur.

3.2.2 Class diagram: A class diagram is a type of static structure UML diagram that describes the structure of a system by showing the system's classes, their attributes, and the relationships between the classes. The messages and classes identified through the development of the sequence diagrams can serve as input to the automatic generation of the global class diagram of the system.

4. DESIGNING CONCEPTS

Defining objects, creating class diagram from conceptual diagram: Usually map entity to class. Identifying attributes.

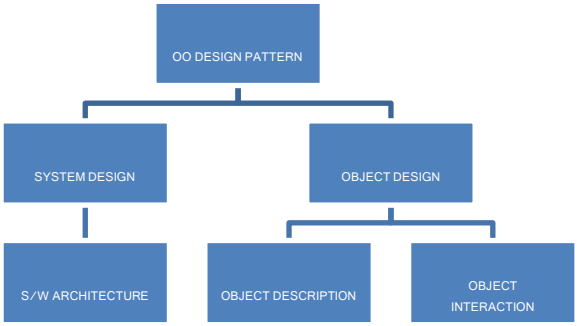

Fig2: Object oriented design

4.1 Use design patterns (if applicable) A design pattern is not a finished design, it is a description of a solution to a common problem, in a context [1]. The main advantage of using a design pattern is that it can be reused in multiple applications. It can also be thought of as a template for how to solve a problem that can be used in many different situations and/or applications. Object-oriented design patterns typically show relationships and interactions between classes or objects, without specifying the final application classes or objects that are involved.

4.2Define application framework (if applicable): Application framework is a term usually used to refer to a set of libraries or classes that are used to implement the standard structure of an application for a specific operating system. By bundling a large amount of reusable code into a framework, much time is saved for the developer, since he/she is saved the task of rewriting large amounts of standard code for each new application that is developed.

4.3 Identify persistent objects/data (if applicable): Identify objects that have to last longer than a single runtime of the application. If a relational design the object relation mapping. Identify and define remote objects (if applicable).

5.DESIGNPRINCIPLESAND STRATEGIES

5.1 Dependency injection: The basic idea is that if an object depends upon having an instance of some other object then the needed object is "injected" into the dependent object; for example, being passed a database connection as an argument to the constructor instead of creating one internally.

5.2Acyclic dependencies principle: The dependency graph of packages or components should have no cycles. This is also referred to as having a directed acyclic graph[2]. For example, package C depends on package B, which depends on package A. If package A also depended on package C, then you would have a cycle.

5.3Composite reuse principle: Favor polymorphic composition of objects over inheritance [1].

6. BENEFITS OF OOAD

Many benefits are cited for OOAD, often to an unrealistic degree. Some of these potential benefits are:

6.1 Faster Development: OOAD has long been touted as leading to faster development. Many of the claims of potentially reduced development time are correct in principle, if a bit overstated.

6.2 Reuse of previous work: This is the benefit cited most commonly in literature, particularly in business periodicals. OOAD produces software modules that can be plugged into one another, which allows creation of new programs. However, such reuse does not come easily. It takes planning and investment.

6.3 Increased Quality: Increases in quality are largely a by-product of this program reuse. If 90% of a new application consists of proven, existing components, then only the remaining 10% of the code has to be tested from scratch. That observation implies an order-of-magnitude reduction in defects.

6.4 Modular Architecture: Object-oriented systems have a natural structure for modular design: objects, subsystems, framework, and so on. Thus, OOD systems are easier to modify. OOD systems can be altered in fundamental ways without ever breaking up since changes are neatly encapsulated. However, nothing in OOD guarantees or requires that the code produced will be modular. The same level of care in design and implementation is required to produce a modular structure in OOD, as it is for any form of software development.

6.5Client/Server Applications: By their very nature, client/server applications involve transmission of messages back and forth over a network, and the object-message paradigm of OOAD meshes well with the physical and conceptual architecture of client/server applications.

6.6 Better Mapping to the Problem Domain: This is a clear winner for OOAD, particularly when the project maps to the real world. Whether objects represent customers, machinery, banks, sensors or pieces of paper, they can provide a clean, self-contained implication which fits naturally into human thought processes.

6.7 Maintainable: OOP methods make code more maintainable. Identifying the source of errors becomes easier because objects are self-contained (encapsulation). The principles of good OOP design contribute to an application's maintainability.

6.8 Reusable: Because objects contain both data and functions that act on data, objects can be thought of as self-contained "boxes" (encapsulation).This feature makes it easy to reuse code

in new systems. Messages provide a predefined interface to an object's data and functionality. If you know this interface, you can make use on an object many context you want. OOP languages, such as C# and VB.Net, make it easy to expand on the functionality of these "boxes" (polymorphism and inheritance), even if you don't know much about their implementation (again, encapsulation).

6.9 Scalable: Object oriented applications are more scalable then their structure programming roots. As an object's interface provides a roadmap for reusing the object in new software, it also provides you with all the information you need to replace the object without affecting other code. This makes it easy to replace old and aging code with faster algorithms and newer technology.

7. DISADVANTAGE OF OOAD

The challenges of OOP exists mainly in the conversion of legacy systems that are built in structured programming languages. The technical challenge is not as big as the actual design challenge.The goal when converting is to minimize the effect the stuctural systems on the OO nature of the new design, and this can sometimes be difficult.

8. CONCLUSION

Object oriented analysis and design elaborates the analysis models to produce implementataion specifications .ooad focuses on what the system does, ood on how the system does. This paper describes it serve as input and output of object oriented design. Analysis and design may occur in parallel in parallel and in practice the results of one activity can feed.Both these process can be performed incrementally. Benefits of ooad often to an unrealistic degree.

9. REFERENCES

[1] Grady Booch."Object-oriented Analysis and Design with Applications,3rdedition":http://www.informit.com/store/product.aspx?isbn=020189551X Addison-Wesley 2007.

[2] Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley. 1995. ISBN 0-201-63361-2.

[3] What Is Object-Oriented Design?. Object Mentor. http://www.objectmentor.com/omSolutions/oops_what.html. Retrieved 2007-07-03.

[4] Harmon, Paul and David A. Taylor. Objects in Action: Commercial Applications of Object-Oriented Technology, Addision-Wesely Publishing, Reading, MA, 1993.

[5] Hayes, Frank. "The Reality of Object Reuse," Computer World, May 6, 1996, p. 62.

[6] Love, Tom. "Seven Deadly Sins of Object-Oriented Development," Journal of Information Systems Management, Summer1995,pp.84-86.

[7] Lucas, Henry. Managing Information Services, McMillan Publishing Company, New York, NY, 1989.