Chapter 3

Object Oriented Paradigm

Object oriented technology (OOT) offers methodologies, techniques, and tools that provide clear advantages for corporate software developers and users alike. OOTs enable fast and cost effective application development with reusable components, provide connectivity across heterogeneous computing platforms, and support integration of data, applications, and business processes. As such, OOTs have been heralded as the next generation of software process technologies beyond structured methodologies, 3GL/4GL, and relational database [1].

We hear the term object orientation quite often in the context of object technology. Object orientation was first introduced in late 1960’s among the researchers. It was popularized in the corporate world in the late 1980s, it became the de facto development standard in mid 1990s. Object orientation is used for modeling a system [3]. This can be applied to various aspects of a computer system, such as analysis and design, models, programs, test cases, databases, architecture, etc. Object orientation is a completely different and novel way of thinking about a system as compared to the structured paradigm.

Structured paradigm is the development strategy based on the concept that a system should be separated into two parts: data and functionality. On the other hand, the main concept behind object oriented paradigm is that instead of defining systems as two separate parts (data and functionality) the system can be defined as a collection of interacting objects [2].
Chapter 3: Object Oriented Paradigm

Requirement Analysis

User Requirement Specification

Software Requirement Specification

Identify Object/Classes

System Test Specification

Establish Object Class interaction

Class Design

Aggregation and Generalization

Analyze Class Hierarchy

Application and Schema Design

Class Implementation

Class Testing

Partial Integration

Class Integration

Build Application

Integration Testing

System Testing

Figure 3.1 An Object Oriented Development Model
Object oriented development starts with object oriented analysis, in which the problem domain is modeled as a collection of objects. During design and implementation, these objects are transformed into other objects of slightly different shape that make up the actual computerized system [4]. An object contains data and operation that operate on that data. Object cooperates by calling operations in each other that is by sending messages. The data in an object is accessible only through operations; no object can access another object data directly. In a true object oriented system, data and functions exists only in the form of objects. Figure 3.1 shows an object oriented development model.

3.1 Traditional System versus Object Oriented System

There are many differences between object-oriented development and traditional, procedural development. Procedural development is usually pursued by techniques such as functional decomposition with stepwise refinement, where the system being developed is described in terms of what it does. Object oriented development does not so much describe what’s being done, rather it focuses on what it is being done to. The expected advantages of object oriented systems relative to traditional systems include[5][6][7]:

- A development processes understandable to users in that the objects model the real world.
- A seamless transition from analysis to final implementation.
- Easier maintenance as objects are modified without changing the entire system.
- Reuse of object in other applications, because objects encapsulate data and provide a service useful in other contexts.
- In object oriented analysis and design there is no H-I-P-O hierarchies - in fact there is no “main”, or topmost object as there is a “main” or topmost procedure in structured techniques.
- Object oriented development is different from procedural development, and requires different skills and way of thinking.
- Through the use of inheritance, polymorphism and dynamic binding, object orientation provides mechanisms, which support software reuse and extensibility as never before. This helps reduce development time, enables systems to adapt more readily to the changing requirements and cuts maintenance bills significantly.
- The support of extensibility also makes object orientation particularly useful in areas where the software requirements are not well understood.
- The incremental development cycle promotes evaluation and re-evaluation from analysis through implementation in a short cycle, enabling the product requirements to evolve along with the software product under development.

The benefits of object oriented development come from careful planning, timely preparation, and easing into this new world in a controlled manner, and by giving thoughtful consideration to the recommendation of those who have blazed the trial earlier.
3.2 Potential Benefits of Object Orientation

Object orientation offers following potential benefits:

3.2.1 Increased Reusability

The Object oriented paradigm provides opportunities for reuse through the concept of inheritance, polymorphism, encapsulation, modularity, coupling and cohesion. The design and development reusability and inter-application reusability through the class hierarchy, which is a direct product of the design process and supported through implementation using an object-oriented programming language.

Although an object oriented approach encourages reuse, there is no doubt that reuse is as much a cultural issue as a technical one. The right management framework and rewards system is required to make reuse work.

3.2.2 Increased Extensibility

Inheritance support extends design in several ways. First, inheritance facilitates reuse to ease the development of new definitions. Second, the polymorphic property of the typing system in object oriented programming languages support extensible designs. Thus, instances of new classes can be introduced and handled using parameters in an identical manner to existing object type.

3.2.3 Improved Quality

Quality systems are on time, on budget, and meet or exceed the expectation of their users. To achieve certain quality criteria are set. Quality criteria are designed checks, the main purpose of which is to ensure that some aspect of a technique or process has been adhered to, or to assess the effectiveness of a deliverable. Quality criteria like extensibility, coupling, cohesion, completeness, performance, security and availability can be assured with the help of certain case tools.

3.2.4 Financial Benefits

Reusability, extensibility and improved quality are all technical benefits object orientation enables to build system better, faster and cheaper.

3.2.5 Increased Chance Of Project Success

A project is considered a success when it is on time, on budget, and meets the needs of its user. Object orientation enables to do all these things and more.
3.2.6 *Reduced Maintenance Burden And Application Backlog*

Software organizations currently spend significant resources maintaining and operating software, and because of the list of work to be done, it takes significant time to get new project started. These two problems are respectively called “the maintenance burden” and “the application backlog”. These are the problems which object orientation can help overcome.

3.2.7 *Managed Complexity*

Object orientation help to manage software complexity by
- Building complex software from well designed reusable objects.
- Designing the software with the expectation that it will need to be modified.
- Responding quickly to the changing environment.

### 3.3 Object Oriented Concepts

Table 3.1 summarizes the object-oriented concepts and terms.

<table>
<thead>
<tr>
<th>Concept</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract Class</td>
<td>A class that does not have objects instantiated from it.</td>
</tr>
<tr>
<td>Abstraction</td>
<td>The essential characteristics of an item, such as a class or components</td>
</tr>
<tr>
<td>Aggregation</td>
<td>Represents “is part of” relationships between two classes or operations</td>
</tr>
<tr>
<td>Association</td>
<td>Objects are related to other objects.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Something a class know(data/information)</td>
</tr>
<tr>
<td>Class</td>
<td>A software abstraction of similar object, a template from which objects are created.</td>
</tr>
<tr>
<td>Collaboration</td>
<td>Classes work together to fulfill their responsibilities.</td>
</tr>
<tr>
<td>Component</td>
<td>A cohesive unit of functionality that can be independently developed, delivered, and composed with other components to build a larger unit.</td>
</tr>
<tr>
<td>Composition</td>
<td>A strong form of aggregation in which the “whole” is completely responsible for its parts and each “part” object is only associated to the one “whole” object.</td>
</tr>
<tr>
<td>Concrete class</td>
<td>A class that has objects instantiated from it.</td>
</tr>
<tr>
<td>Encapsulation</td>
<td>The grouping of related concepts into one item, such as a class or a component.</td>
</tr>
<tr>
<td>Information hiding</td>
<td>The restriction of external access to attributes.</td>
</tr>
<tr>
<td>Inheritance</td>
<td>Represents “is a” and “is like” relationships.</td>
</tr>
<tr>
<td>Instance</td>
<td>An object is an instance of a class.</td>
</tr>
<tr>
<td>Interface</td>
<td>The definition of a collection of one or more operation signatures that defines a cohesive set of behaviors.</td>
</tr>
<tr>
<td>Message</td>
<td>A message is either a request for information or a request to perform an action.</td>
</tr>
<tr>
<td>---------</td>
<td>--------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Method</td>
<td>Something a class does (similar to a function in structured programming)</td>
</tr>
<tr>
<td>Object</td>
<td>A person, place, thing, event, concept, screen, or report.</td>
</tr>
<tr>
<td>Override</td>
<td>Sometimes you need to override (redefine) attributes and/or methods in subclasses.</td>
</tr>
<tr>
<td>Persistence</td>
<td>The issue of how objects are permanently stored.</td>
</tr>
<tr>
<td>Polymorphism</td>
<td>Different objects can respond to the same message in different ways, enabling objects to interact with one another without knowing their exact type.</td>
</tr>
<tr>
<td>Subclass</td>
<td>If class B inherits from class A, we say B is a subclass of A.</td>
</tr>
<tr>
<td>Superclass</td>
<td>If class B inherits from class A, we say A is a Superclass of B.</td>
</tr>
<tr>
<td>Transitory object</td>
<td>An object that is not saved to permanent storage.</td>
</tr>
</tbody>
</table>

### 3.4 Object Oriented Analysis

Analysis focuses on producing a model of the system, called the analysis model, which is correct, complete, consistent, and verifiable [9].

Object oriented analysis can be performed on two levels [5]. At highest level it can be used to analyze large domains such as a complete industry, a complete enterprise, a major business unit, a group sharing common concepts and a subset of organization’s activities. At this level the objective is to understand an enterprise and the domain in which it exists. It is concerned with understanding the motivation behind, and planning to provide a set of systems solutions within that domain. At next level, analysis seeks to obtain a thorough understanding of a specific problem domain, by representing the real world as a collection of intercommunicating objects.

The process of analysis is iterative and involves the following stages:
- Quantify the business objectives. This will produce a list of system requirements to achieve the business objectives.
- Identify candidate classes in the problem domain. Language analysis can be very effective: plural nouns nearly always identify classes.
- Identify the properties of classes-responsibilities and collaboration with other classes. Words and phrases meaning possession indicate properties. Verb indicates operations.
- Identify the methods of the classes. Many of these are exposed by thinking through the various interactions the system must support, that is, the dynamics of the scenario.
- Develop the life cycle stages for each identified class.
- Identify the user interfaces to the system.
- Rationalize the classes found so far. This entails factoring out the common essential characteristics among classes and promoting general classes.
Chapter 3: Object Oriented Paradigm

The analysis model is composed of three individual models: the functional model, represented by use cases and scenarios, the analysis object model, represented by the class and object diagrams, and the dynamic model represented by statechart and sequence diagrams.

Object-oriented analysis (OOA) is concerned with developing software engineering requirements and specifications that expressed as a system's object model (which is composed of a population of interacting objects), as opposed to the traditional data or functional views of systems. OOA can yield the following benefits: maintainability through simplified mapping to the real world, which provides for less analysis effort, less complexity in system design, and easier verification by the user; reusability of the analysis artifacts which saves time and costs; and depending on the analysis method and programming language, productivity gains through direct mapping to features of Object-Oriented Programming Languages.

3.5 Object Oriented Design

Object oriented design (OOD) builds on the products developed during Object-Oriented Analysis (OOA) by refining candidate objects into classes, defining message protocols for all objects, defining data structures and procedures, and mapping these into an object-oriented programming language (OOPL). Several OOD methods (Booch, Shlaer-Mellor, Buhr, Rumbaugh) describe these operations on objects, although none is an accepted industry standard. Analysis and design are closer to each other in the object-oriented approach than in structured analysis and design. For this reason, similar notations are often used during analysis and the early stages of design. However, OOD requires the specification of concepts nonexistent in analysis, such as the types of the attributes of a class, or the logic of its methods.

OOD has two separate components which are blended in the development of application: class design and application design.

Class design is wrapped under application design. The design of an application leads on from analysis which identifies the kinds of entities involved. These entities may be tangible objects as traffic lights, chairs or aeroplanes. They may equally well be abstract concepts such as roles, interactions or incidents. Once these entities are described, application design proceeds by connecting instances of the classes so they interact with each other resulting in a solution to the stated requirements. The main stages in the class design are listed below:

- Identify classes;
- Identify subclasses within each class;
- Identify abstract behavior of each class;
- Identify common behavior (abstraction);
- Identify specific type of behavior (specialization);
The main purpose of class design is to design for reuse and to design in a highly modular way. The process of abstraction and specialization creates better reuse possibilities. Concentrating initially on individually classes rather than the architecture of the end application creates a more modular system. End-application design is a top-down and bottom-up process, designing the application from the existing building blocks. Additional classes are required as a result of over all architectural issues.

3.6 Object Oriented Programming

The history of object oriented programming languages (OOPL) can be traced back to the early 1960s, and to a computer simulation language called Simula. Since then various object-oriented languages like Smalltalk, Eiffel, C++, Java etc evolved, some being purely object oriented and some being hybrid. The term object oriented programming language means different things to different people and there is varying degrees of object orientedness. Wegner classifies these degrees as follow [5]:

- **Object-based languages**: the set of all languages that support objects.
- **Class-based languages**: the subset that requires all objects to belong to a class
- **Object Oriented languages**: the subset that requires classes to support inheritance.

One way in which some languages (pure OOPLs) enforce the use of object-oriented behavior is by requiring that all functions, procedures and variables be declared internal to some class. In such a system there are no global variables, and no freestanding functions or procedures. Objects encapsulate all states, and all code executes relative to some specific object. The main program becomes the constructor method in the class that defines a specific application. All OOPLs support the basic concepts of objects, classes, message passing, inheritance and polymorphism to some degree or other. However, some languages support a class being an instance of a higher order class called a metaclass. In most of the OOPLs classes are factories that create and initialize instances.

Object oriented analysis and design models are transformed into object oriented source code. Transformations enable us to improve specific aspects of the object design model and to convert it into source code. By providing systematic recipes for recurring situations, transformations enable us to reduce the amount of effort and the overall number of errors in the source code. However, to retain this benefit throughout the lifetime of the system, it is necessary to document the application of transformation so that they can be consistently reapplied in the event of changes to the object design model or the source code. The transformations should be applied such that it would avoid system degradation. These include [9]:

- Optimizing the class model
- Mapping associations to collections
- Mapping operation contracts to expectations
- Mapping the class model to a storage schema.
To get a high quality source code it is necessary to get the design right first, try to develop smaller methods, reduce method response, develop in small steps and write code that is understandable by everyone.

### 3.7 Object Oriented Testing

Three concepts have major impact on testing approaches when one move from testing traditional systems to object oriented systems. These are information hiding, encapsulation, and inheritance. According to [3] testing object oriented application one should think about three important points, listed below:

- Enhance the definition of testing to include defect-finding techniques that are specific to object technology (i.e. to OOAD and OOP).
- Change the strategy for unit and integration testing to a large extent.
- Pay attention to the specific details of the OO aspects of the software.

Object oriented software projects begin with detailed requirements analysis and design phases that are also object oriented in nature. That is we start we OOA, progress to OOD, and then to OOP. Throughout the lifecycle of the project, the OOA and OOD specifications keep evolving, and get enhanced. This also means that we can test these models as they evolve, rather than waiting for the entire software product to be ready. When reviewing OOA and OOD models, two factors should be taken into consideration:

- Accuracy of the OOA and OOD models
- Consistency of the OOA and OOD models.

Object oriented testing techniques involve:

- **Unit testing**: A unit in OO system is a class. Even if one method in a class is tested it cannot be said that the unit testing has been completed if the class is inherited. It is necessary to test the method in all the classes in the inheritance hierarchy to be confident that it is working fine.

- **Integration testing**: Integration testing in OO systems can be done by using thread-based testing and use-based testing. Thread based testing involves integration of classes needed to respond to an input provide to the system. In use-based testing, a layered –approach is used. The idea is to start with simple test cases and then move towards the more complex ones.

- **System testing**: System testing aims at testing the functionality of the software.

Testing activities are performed throughout the development. Test planning should be done early, as should the identification and specification of the tests.

### 3.8 Object Oriented DBMS

The need to use a database often arises from the limited capacity of primary memory. Therefore databases are often stored on secondary storage, providing efficient ways to access the objects. Another need that arises is the ability to store objects longer than a
program execution. Thus we want the objects to survive the execution that created it. This ability is called persistence.

Relational systems were first introduced in the early 1970s. During the late 1980s they totally dominated the DBMS industry. They serve as a simple and logical view of the data stored in the database. The user views the data as stored in tables, which he or she can manipulate. One important idea is that data be stored in as few places as possible. Different normalization techniques support the process of making the tables as independent of each other as possible.

Simplicity and data independence is the major features of a relational DBMS, but also can be a drawback in some applications. The relational model cannot capture the semantics of complex objects. To model a complex object we often have to split the information into several tables; this makes each access to such object slow since the DBMS must join a lot of tables to gather the object’s information.

The idea of an object DBMS is to store the objects as such, and thus bridge the semantic gap all the way to the databases. In this way we do not have to perform any slow joins to get access to a specific object. By storing the objects as such instead, it is possible to express the semantics of the objects in much better way than possible in relational system [8]. This implies that OODBMS is very useful in applications where complex objects must be persistent. Typical examples are different kinds of design support such as CAD, CASE, CAE and the like.

The major benefits of using OODBMS include:

- Objects as such can be stored into the database.
- No conversion is needed for the OODBMS type systems; the user-define classes are used as types in the OODBMS.
- OODBMS can be integrated with the object oriented programming language. The language may even be exactly the same as that used in the application, which does not force the programmers to have two representations of his objects.

The Object Database management Group (ODMG) has come up with a standard specification for using OODBMS in certain way [3]. This specification has three main aspects:

- The Object Definition language (ODL) is very similar in concept to the Data Definition Language of SQL.
- The Object Query language (OQL) is very similar in concept to the Data Manipulation language (DML) of SQL.
- The Object Manipulation language (OML) is an extension of OQL.

To summarize Object oriented DBMS have been developed to store objects as such in the database. No standard, or even general consensus, exists on what defines an OODBMS. Different application areas have different requirements, and many vendors optimize their products for a specific application area. Using an OODBMS in an
Chapter 3: Object Oriented Paradigm

OOSE development is often easy. Generally, no extensive overhead needed to incorporate the ODBMS in the application being developed.

3.9 Problems with Object Oriented Paradigm

In spite of the claimed benefits the issues presented below forces us to think if object orientation is really taking us for building a quality product.

The claims by many OO methodologist that the transition from OO analysis to OO design is easy and smooth may seem convincing but it is not so. The difficulty in moving from OOA to OOD is caused by the fact that OOA and OOD objects represent inherently different things. As a result, and OOA model cannot simply become an OOD model [10]. Hermann Kaindl has very effectively brought forth the difficulties encountered in transition from OOA to OOD.

The naturalness claim made by the advocates of the OO paradigm is based on two fundamental assumptions:

1. Assume that the world is made of objects and relationships.
2. Assume human beings think in terms of objects and classes, etc

Detienne [14] shows that in fact the naturalness claim of OO may not be entirely accurate. This would then appear to contradict the claim made by OO advocates. In arguing against some of the claims made in OO paradigm, Detienne states:

"It appears from our study that designing a problem with an OO languages is not so easy and so natural as the advocates of OOP say. The hypothesis on naturalness of design with an OOP language is not confirmed by our data. Our observations show that decomposing a problem into solution is not easy, especially for identifying classes and attributing objects and functions…"

Mc Ginnes [13], argues that objects in the real world are distant from those in the information system. Often reality needs to be “bent” around the OO model. The lack of intermediate model between the real world and the information system can be dangerous and result in misuse.

Ling [12] points out that OO model, doesn’t provide adequate support for views. View support is also essential for other issues e.g., security, corporate databases etc.

According to Date [11], it leaves data analyst with two key dilemmas:
1. In the pure OO model ad-hoc queries are not possible( as methods need to be predefined)
2. As a result of (1), if methods are the only way of interrogating an object (i.e., performing some operations on it), this then implies that in OO databases, and every query must be predefined.

Advocates of the OO paradigm, frequently claim that OO approach to software development allows software to be extended, and modified easily at later stages. Lubars et al [15], reports some extensibility problems in the development of an object
oriented specification for an automated teller machine. They concluded that extensibility needs anticipated in advance and objects structured appropriately. They however complain that current OO approaches rely on simple guidelines and strategies; they point out that “reliance on simple guidelines and strategies from methods textbooks may… cause the analyst to miss reuse opportunities and make the model more difficult to change.”

There are several myths [16] encountered related to object oriented paradigm. It therefore becomes essential to concentrate on the facts in order to gain a proper insight to the given problem, which is to be solved object oriented. Then it will be possible to gain true benefits of object orientation.

Thus we see that most of the claimed benefits of OO paradigm can be challenged, then were lies the quality of the product when developed using OO paradigm. This indeed requires more efforts to make OO approaches truly reliable. Although this is not a task of day or so or a task that can be managed and worked out by a individual or a group it will take time to reap the fruits of object orientation.

Because object-oriented design has a programming outcome in mind, the results of the design are easier to code than general requirements. When object-orientation enhances understanding, it increases the option of reuse. Also, parts of projects become more meaningful in themselves and are more likely to be reused. The message-driven nature of objects makes it easier to bring the object into another project because the object has its own methods that it uses on its own data. The methods become active in response to program messages, requests, or events. Enhanced understanding also makes maintenance easier. Because the design process is emphasized and objects have a more understandable definition than groups of functions, it's easier to know the status of a project and when a project is complete. It may be easier to see where functionality should be added or to what objects functionality could be added. There are blurred boundaries between object-oriented design and both object oriented analysis and implementation. Also, database schema is based on the object design model, not on the source code. To tackle the myths and problems related to object orientation object oriented design is the better phase. For this and many reasons this work emphasizes on object oriented design phase in detail to predicting the reliability of the object oriented software product.

References