International Journal of Computer Science and Applications,
Vol. 5, No. 3, pp 9 - 22

EXPLAINING OBJECT ORIENTED ANALYSIS (OOA) CONCEPTS TO MANAGERS OF AN ORGANIZATION

K. V. DINESHA, P. G. BHAT

Indian Institute of Information Technology,
Bangalore, India
dinesha@iiitb.ac.in, bhat.pg@gmail.com

It is said that objects occur naturally, meaning that good Object Oriented (OO) model reflects reality. The person who best understands reality is the domain manager; unfortunately he is not computer savvy. This paper attempts to make the manager aware of fundamentals of OO thinking so that he/she feels confident of attempting a first cut OO model of the intended application. Every manager worries about what the proposed software should do for each of the stakeholders for whom the software is being developed. Experiments in industry have brought out that teaching domain specialists about use cases and object concepts make them enthusiastic about participating in the OO modeling of the application. All that is necessary is to expose OO thinking in a way that it reinforces their approach to handling complex situations in practice. This paper is an evangelistic attempt to make managers adapt to OO thinking while conceiving new applications, through easy to understand approach to concepts such as Objects and Use Cases.

Keywords: Object Oriented Analysis, managers of end user departments, Objects, Use cases, Scenarios, Flow of Events, Boundary, Control and Entity objects, Objects interacting with each other, association, aggregation, composition, and inheritance.

1. INTRODUCTION

Several OOA concepts are already used by the customers while analyzing, designing, and implementing the 'non-software version' or 'manual version' of their systems. The OO way of analyzing and designing a system emanates from designs and procedures to build large systems. Hence the Object Oriented thinking starts from the requirement phase. The OO concepts help us to think and communicate with the customers and end users.

A high level model for the people involved in software development process is represented in fig.1 below.

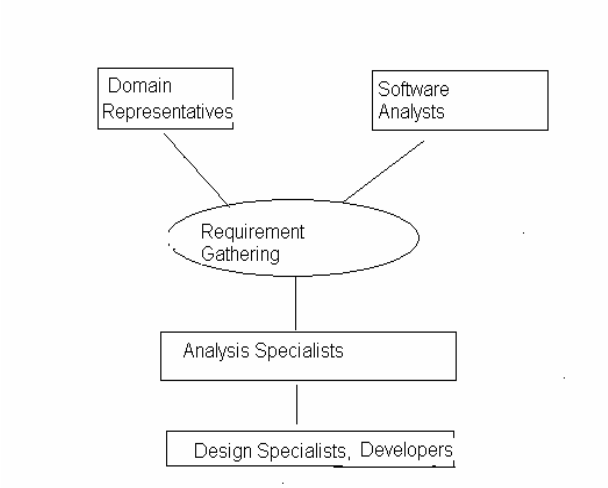

Fig. 1

Usually the managers from the end user departments constitute the domain representatives and software analysts constitute the software company representatives.
The domain representatives should be able to communicate with the software analysts regarding their requirements with the confidence that a software system can be built as per their needs. The understanding that the concepts used by them in building, managing and maintaining manual systems also form the basis for building software system will hopefully give the required confidence and comfort level for the domain people while participating in gathering requirements along with software professional. In this article we attempt to give such an explanation.
We explain the following concepts:

- Modeling of an object
- Object as a unit of organization for a system
- Service (use cases) and actor view of a system
- Concept of scenarios, flow of events
- Objects interacting with each other to deliver a service
- Boundary, Control and Entity objects
- Relationship between objects (dependency, association, aggregation and composition)
- Inheritance

In the last section, we have given the details of the 'feedback and experience from the participants'.
Our audience is the managers from the departments of end user organization. We also assume that the systems are conceived to be implemented as manual system. Conceptually, implementation details are irrelevant; however it is needed to reinforce the idea that OO concepts existed and were used extensively in modeling organizations, systems and procedures in business long before the use of computers in business.
The concepts discussed in this article are standard OO concepts. Many of these concepts are explained from end user perspective in bits and pieces in many places but in different

contexts. However, to the best of our knowledge, all these concepts are not discussed together in one place with a single theme, purely from the perspective of end user Managers. The examples, style, and framework are evolved over a period while teaching these concepts to different types of audiences. As these concepts are standard, we have not indicated the references at individual parts. The references are given at the end.

2. MODELING OF AN OBJECT

We model every thing that we interact with. A simple example is a pen. We associate writing (the purpose) when we think of pen. Of course, subconsciously we also associate some shape, weight, color (attributes) etc. We associate a state to the pen like it is filled, half filled or empty etc. In general, we model a pen for (a) its behavior (purpose : I have a teacher friend. He had a pen exclusively used for marking students' answer scripts and since most of the students failed in his paper, he calls that pen as The Cruel Pen) and (b) its attributes. We even go one step ahead – we equate a pen with 'its model'. However, the model and the object are different. For different types of customers (user) the same pen is modeled differently. My four year old son firmly believes that a pen is used only to pierce papers. That means depending on the application context, the same pen is modeled (with its attributes and behaviors) differently. Interestingly this leads to a conceptual advantage for an object.

The concept of object helps to integrate systems around similar concepts. Consider an example of a student being modeled in different application domains like examination system, registration system, and sports management system. It is the same student modeled differently in different systems.

The concept of objects also helps us to achieve a first level of reuse, For example, if we have analyzed the 'Examination System' of an university, and found objects like Student, Professor, Department, and Courses, we can expect these objects to be present in other related systems like registration system, university sports system etc. Of course, there will be extra attributes and methods.

3. OBJECT AS A UNIT OF ORGANIZATION FOR A SYSTEM

Consider the administrative department of a company which is implemented mostly manual. The admin dept has officers in-charge of students information, students' fee collection, faculty information, leave record management, hostel related functions etc. The dept has, say, five people. As a first step in organizing his office, the admin manager would group the functionalities into meaningful groups like student related activities, faculty related activities, leave maintenance related activity etc and assign a person responsible for each of these activities . This person would also be in charge of the respective files and records i.e., the respective data. We will not find a function or data with no one responsible. Also, one function/data is owned by only one person. To summarize, the functions and data of the office is organized into units called objects where each object performs a set of functions and owns the corresponding data. We will not find a function/data being shared by two objects or a function/data being orphans (does not belonging to any object). Thus well-defined objects with responsibility for functions and ownership data form the basic unit of organization of any system.

4. SERVICE (USE CASE) AND ACTOR VIEW OF A SYSTEM

A system can be seen as collection of 'services' offered to its 'users'. Let us take a few examples in every day life:

(a) We go to an airline travel agent with a request to book a ticket, enquire about availability of a ticket, cancel a ticket etc. The travel agency offers the 'service' of booking, enquiry, and cancellation to its customers ('users'). Similar cases arise when we do a booking of railway ticket, bus ticket etc.

(b) We plan our vacation. We go to a travel agency, which gives various options. The options are different possible routes, different duration(3 days, 5 days etc), different accommodation types, food types etc. The agency will create a package which suites our needs. Different packages form different services by the agency.

(c) Work flow management of a hospital clinic. The patient registers first at the registration counter, then goes to a 'first level' doctor who does certain preliminary check ups and, if needed, advices a few tests like blood, urine etc. The test reports and assessment of 'first level' doctor will go to the specialist. He will then examine the patient and suggest some more tests or prescribe treatment.

At the highest level, we model a system as follows:

A user interacts with a system with the intention of *getting a benefit/service*. The system provides a set of promised *services and only those services*.

First identify all types of users and the respective use cases Let us first understand the interpretation of the term '*Use case*'.

(1) Use case can be a benefit/service given by the system to a user: For example, in an ATM system of a bank, the customer wants to
   i. withdraw money,
   ii. get a list of past transactions
   iii. transfer money from one account to another etc.

(2) Use case can be a feature to prevent a user from doing something. In the ATM, if a thief comes and bangs the ATM, the system should alert the nearest police station(or incite a trained dog to catch the neck of the person)

(3) Use case can be a set of inputs for the system and/or set of outputs/reports. For example, in the ATM case, the input could be the account number / password entered by the customer. The output could be the list of latest ten transactions.

In all the above cases we do not worry about the 'implementation' of these use cases. In the ATM case the implementation could be a complex network connections with distributed databases or it could be the following implementation:

Inside the ATM box, a person is sitting and keys a set of word files which will be displayed on the screen. This word file looks *exactly* like that of normal ATM. When the user introduces the card, the person 'pulls' the card with exactly same force of a real ATM and checks from a book in his hand regarding the availability of the balance, authenticity of the user(the person is gifted with the ability to 'read' a magnetic tape) and at the end takes dollar notes(making the same 'karrr..' sound as an ATM would do) and push it through the money slot of the ATM.

The above implementation *does not change* the use case. A software system may not implement the use case also (for example the use case number 2 listed above might be implemented by some hardware systems or by a set of trained dogs).

In general, we call these use cases as *Business use cases*. However, if we build a software system to implement the use case, then we call it a *system use case.* The beneficiaries or the users are called *Actors*.

Step 1 *Identify the use cases and the respective actors of the system.*

Refining use cases is not a one step process but an iterative one.

In the *first iteration*, we start with a '*candidate list of use cases'*. Like for ATM system, the candidate use cases could be

- withdraw money
- list previous transactions
- login to system,
- deposit money in ATM etc.

When identifying a use case, we should note that it must be *complete* and gives *end to end service* to the actor. For example,

> If we implement only 'login', what benefit will the customer get(say, after 'login', when he wants to withdraw money, the ATM responds 'sorry, system under construction'). Login *alone* is not a benefit.
>
> If the service is 'withdrawal of money' it should include the service 'inform the bank about the details of the withdrawal'. (this information helps the bank officials to know the status of the cash in the ATM and at appropriate time, the bank sends designated person to fill the ATM with cash). What is the use of an ATM which after sometime do not have money in it?

When we use the criteria of 'completeness' and 'end to end service' for identifying the 'size' of a use case, we get use cases which are reasonably independent and can be developed by different teams.

In the *second iteration* of arriving at use cases, we look for any part which is common to some/all of the use cases. 'login part' is an example. We separate them out and make them independent use case. The advantage of this is that the common part need not be developed by many teams and perhaps a specialized team can develop it as it needs some specific expertise and it needs to be thoroughly tested as it impacts many use cases.

Step 2 : *For each use case / actor scenario obtain the 'Flow of Events(FOE)'*

The use cases and respective actors give at a high level the specification for the interaction of the system with its beneficiaries(actors). However, this is not a complete description of the interaction. To make this description complete, we need to go one more level of detail. This level of detail is FOE. To understand the concept of FOE, we consider *a scenario* for a use case/actor pair.

As an illustration, for the ATM case, assume that the ATM is implemented. Consider a scenario where 20 customers c01, c02, …, c20 are standing in a line before the ATM. Assume that all these customers want to withdraw money from their savings account. Each customer enters into the ATM cabin and interacts with the ATM to withdraw money. Imagine that there is an 'invisible person' inside the ATM cabin and notes down the 'interaction'(of course he does not 'interfere' with the interaction and hence will not affect the outcome of the interaction) between the ATM and the customer.

The following is the story as recorded by the 'invisible person' of the interaction of different customers.

Story of customer c01 :

ATM : insert your card	C01 : Inserted card
ATM : enter your user id and Password	C01 : entered id & password
ATM : sorry user id & password pair is wrong .. re-enter	C01 : entered again
ATM : enter amount to be Withdrawn	C01 : entered the amount
ATM : gave the amount	C01 : took money, happily out

Story of customer c02 :

ATM : insert your card	C02 : Inserted card
ATM : enter your user id and Password	C02 : entered id & password
ATM : sorry user id & password pair is wrong .. re-enter	C02 : entered again
[This happened 3 times, ATM grabbed the card and told c02 to contact bank office ..]	

Story of customer c03 :

ATM : insert your card	C03 : Inserted card
ATM : enter your user id and Password	C03 : entered id & password
ATM : enter amount to be Withdrawn	C03 : entered the amount
ATM : Sorry there is insufficient money in your account	

Question : by going through stories of about 10 customers, could you get a pattern and fit the interaction between ATM and 11th customer ?

Answer : It is possible and the general pattern looks like this:

The ATM asks the customer to insert the card.

If the card is not properly inserted, ATM repeats the request to insert the card till it can read the card.

ATM asks the customer to enter user id & password

If the user id & password pair is wrong, it asks him to re-enter. If the pair entered is wrong for 3 consecutive times, the ATM grabs the card and asks the customer to call number …

ATM asks the customer to enter the amount to be withdrawn.

If the amount is less than balance, the customer gets the amount

Otherwise, a message that insufficient balance will appear.

The above template is not the interaction of a specific customer, however, it is the 'logic' behind the interaction of most of the customers.

This description of the interaction of an actor with the system is called as 'Flow of Events' associated with the use case actor pair for a scenario.

5. OBJECTS INTERACTING WITH EACH OTHER TO DELIVER A SERVICE

In the previous sections we discussed the concept of 'objects being the basic unit of organization for a system' and 'use cases' capturing the behavior of the system. We now try to understand, how a system realizes a use case for an actor. To understand this we have to look *inside* the system.

The key principle here is that 'objects exchange messages with each other to deliver a service to an actor'. To understand this, consider the example of a railway reservation system. This system offers the services of booking a ticket to a train, canceling a ticket etc. Let us also assume that the system is implemented when no computers were available, i.e. entirely manual. Fig 2 below gives the organization of the system.

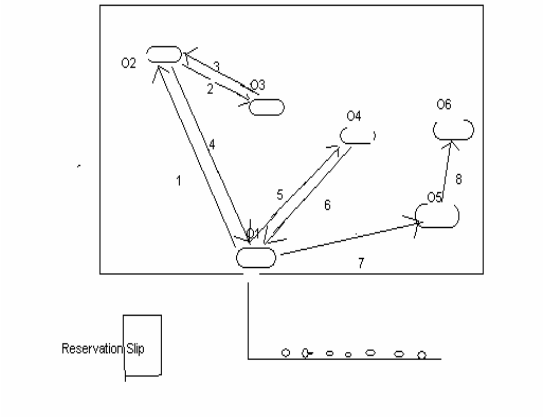

Fig 2: Railway Reservation Office
(Big Rectangular Box)

The objects *inside* the Office are

O1 : The officer at the booking counter. He is the interface of the system to the actors

O2 : The officer in charge of Delhi bound trains.

O3 : The officer in charge of Chennai bound trains

O4 : The officer in charge of computing the train fares.

O5 : The officer in charge of Communicating to train system

O6 : person responsible to preserve the reservation slips

Let us assume that the first customer wants a ticket from Bangalore to Delhi via Chennai in train number 1423 on 14 May 2005 for 3 passengers. First he fills the reservation slip with relevant details and gives it to the booking officer O1. This request is serviced by the system in the following way.

The officer O1, looking at the destination as Delhi, sends the slip to the officer(object) O2 [asking him to tell, if the ticket is available or not, if available what are the seat/birth numbers etc]

… *Message 1* in Fig 2.

O2, looking at the slip, finds that the journey is via Chennai and so sends the slip to O3[asking him, if ticket to Chennai is available etc].

… *Message 2* in Fig 2

O3 checks his register for that date(assume ticket from Bangalore to Chennai is available) enters the seat/birth numbers on the slip and also makes relevant entries in his register, and sends the slip back to O2.

… *Message 3* in Fig 2

O2 enters the details related to Chennai – Delhi journey (assuming that the seat/birth is available) and sends the slip to O 1.

… *Message 4* in Fig 2

O1 needs to know the total fare. So he sends the slip to the fare computing specialist O4 requesting to compute the fare and write on the slip.

… *Message 5* in Fig 2.

O4 computes the fare and writes on the slip and sends to O1.

… *Message 6* in Fig 2.

O1 takes the money from the customer and give the ticket. However, the transaction is not complete. When the passenger go to the railway platform on 14/05/2005, he should have his name on the train coach. This part is the responsibility of the officer O5.
O1 sends the slip to the officer O5 requesting him to make necessary entry in his ledger.

… *Message 7* in Fig 2.

O5 after making the necessary entries sends the slip to O6 who preserves reservation slip (say for 2 years as per the law of the land).

… *Message 8* in Fig 2.

So, essentially, the object O1, O2, …, O6 inside the system, by exchanging the messages 1, 2, ..,8 between themselves, delivered the service ' booking a ticket from Bangalore to Delhi for ….. '. This is one of the foundation principles of OOA.
Notice that initially, when we started analyzing the system, we observed *reservation slip* as an object and routinely preserved, as our analysis instinct said that all objects need to be identified. One advantage of identifying this object is that it can be given multiple implementations (example in different languages, by different vendors etc). Also, we see that at the end, *inside* the system, it will be preserved. Interestingly, this object can be a link object to another system like insurance claim settlement system (this system will be invoked, for settlement purpose, in case of any accident). The beauty of OO analysis methodology is that we identified a useful unit of the system, without knowing about the system details or we identifying a building block of the system without knowing how it is a building block.
Let us look a little more into this sequence of exchange of messages 1,2,..,8.
The object O1 sends message 1 to object O2, *because object O2 could give that service to the object O1*. Which means, message 1 is one of the services offered by O1. By similar observation, we conclude that
Messages 4, 6 form the basis of services offered by the object O1. Similarly messages [1,3] for O2, messages 2,5 for O3, Message 7 for O5 and message 8 for O6.

The sequence of steps to deliver the service can be symbolically represented as below.

1. The object O1 invoked the service O2.message1.
2. The object O2 invoked the service O3.message2.

3. Similarly the objects O3,O2,O1,O4,O1,O5 invoked the services O2.message3, O1.message4, O4.message5, O1.message6, O5.mesage7, O5.message8 respectively one after the other.

[NOTE : the above sequence is also a pseudo code for a java program describing the 'manual' steps]

The system delivers service to an actor by exchange of messages by the objects within the system. This is a language which can be easily understood by the domain person.

If we ask the question, how could the *object O3 deliver the service O3.message2* ? we find that a exchange of sequences of messages within O3 will achieve this. In a nut shell a collection of exchange of messages between the objects ultimately represents the dynamic behavior of the system.

6. BOUNDARY, CONTROL AND ENTITY OBJECTS

Let us consider the 'inside' (manual implementation) of railway reservation system discussed earlier. In this system, there are objects of the type O1(booking officers). This type of objects essentially interface with the outside world i.e., with the actors. We necessarily need this type of objects for most systems. We call them 'boundary objects'. There can be five counters which deal with the reservation work. All these objects receive requests from the actors. Again, there will be some officers(objects like O2(officer in-charge of reservation of Delhi bound trains), O3(officers in-charge of Chennai bound trains), O6(officer in-charge of collecting reservation slips) etc.) These objects *store* some data or are the owners of data of the system. These objects are called *entity* objects. There is one more type of object which is implicitly assumed in the office. To understand this type of object, consider the office organization in our reservation system.. The first customer requests for Delhi reservation. The officer O1 takes the reservation slip. Inside the reservation office, all the objects O1, O2 etc will be sitting peacefully doing their work. We have an office boy whose job is to collect the slips from the officer O1 and looking at the destination, will deliver the slip to O2 or O3 or any other object. Similarly he does this for all objects. Of course, the office boy's job is only that of a *postman*. He cannot *take decision* on behalf of any object (for example, through his experience, he may 'know' that there are no tickets available in April. Still he should pass a slip with reservation request to Delhi for 10/April to the object O2. It is the responsibility of O2 to mark on the slip that 'no seats are available'). This office boy plays the role of '*control object*'. There are many advantages of the control object(office boy). Some time if we add more boundary objects (reservation counters), the people in the new counters need not know about the addresses of all 'entity' objects. The entity objects need not know about the addition of new boundary objects. If for some days, say the officer O2(one of the entity objects) is on leave, and the officer O3 handles Delhi bound trains, the boundary objects need not be worried about these change. The office boy takes care of all these problems and delivers the information to the right party. To summarize, first level implementation of any system involves boundary, control and entity objects. It is useful to interpret that the boundary objects interact with the users to accept inputs and deliver outputs.

7. RELATIONSHIP BETWEEN OBJECTS

(DEPENDENCY, ASSOCIATION, AGGREGATION AND COMPOSITION)

7.1. *Association*

In an object-oriented solution, we find an organisation of objects, each object playing a set of roles and each role handling a set of responsibilities.

Playing a role, a person has some contract with the external world to provide a set of services. A taxicab driver plays his role and is responsible to ply passenger for a fair fixed by the authorities. If a person approaches a taxi in a stand or calls one by radio, the taxi cab driver or taxi company is obliged to transport the person from the origin to a desired destination. To request for the services of a taxicab, we have to associate with the taxi driver by making eye contact or calling him over a phone. We cannot request a service from an entity before establishing a contact, i.e., before forming an *association*.

In the society, an entity implicitly or explicitly announces the services offered by it. An object, which is a model of a real world entity, exposes its interfaces announcing to other objects the services it offers. Viewing differently, the interfaces announce the responsibilities of an object. Another object in the application that can establish an association with this object can send a message (aka request) requesting the object for some services.

Thus, association between two objects is necessary before they can communicate with each other. When an object sends a message to a second and for its own behavior depends on the resulting state of the second object that has acted on the request or the on the value returned, there is a dependency by the first object on the second.

7.2. *Whole-Part Relationship – A Special Type of Association*

In an association, an object seeks services of another object to achieve a result that it could not have achieved on its own. But, the modeling of the object itself is complete. When an object's own definition does not structurally depend on the associated object, but has a reference to the other object, we call the relationship as association.

There are several cases where we build an object by assembling other objects. We form a whole by using different parts, though the parts themselves may be useful as independent objects. A taxicab has parts like steering system, braking system, engine, body, wheels etc. To behave like a taxicab, it needs all the parts. For the cab to move faster, the engine has to accelerate. A passenger asks the driver to move faster or slower; the taxi sends a message through the accelerator pedal to the engine to rotate faster or slower. We don't say that the engine is rotating faster, but that the car is speeding. Seen from outside, we see the whole object, and the parts are hidden. The structure and behavior of the whole depends on the parts. When a behavior is requested, the whole delegates the responsibility to appropriate parts.

The whole is aware of its parts as it requests for different services from the parts. A part need not be aware of the whole. It would respond to the requests from the whole. Being part of the whole, and not being visible to other objects, it receives requests only from the whole whose part it is.

Whole-part relationships are classified as '*aggregation*' and '*composition*' based on how closely the parts relate to the whole.

7.3. *Aggregation*

When the parts are not created for the sake of whole and can be interchanged during the life of the whole, it is aggregation. The parts could be manufactured separately and assembled to create the whole. Parts of an automobile are manufactured separately and are changed during the life of the vehicle. Also, a part may be used with another whole –

a wheel that is used in a car may be used in a bullock cart too(Seven years ago, I went to buy a tyre for my Maruti car. The shop keeper asked if I needed the tyre for a bullock cart. He said that no tax would be levied on this purchase, if it was for a bullock cart).
In some cases, a part may be shared by several whole objects simultaneously. If we consider a sports team with several players, these players may be members of several teams at a time. Several computers may share one monitor. It is possible to use a TV set as a TV receiver and an AV player for DVD.
As the parts are not created for the sake of whole and also may be shared, the whole does not have the responsibility of creating the parts and does not have right to delete the parts. The whole does not own its parts.

7.4 Composition

Composition is a special kind of aggregation.
The tyre of a car is manufactured using steel re-enforcer and rubber. We do not recompose the parts during the use of the tyre nor do we separate the parts after it is manufactured. When we compose a word, we use several characters and when the word is deleted all the characters in the word are also deleted. In composition, the parts are created for the whole and are deleted with the whole.
In a composition relationship, the whole owns its parts; it has the responsibility to create and delete the parts. Hence, the parts cannot be shared with other whole objects.
Model of a whole-part relationship (aggregation Vs. composition) depends on the domain and context. E.g., for an end user, a laptop computer would be composed of its parts and a desktop computer aggregation of its parts. For a manufacturer of the PCs, laptop and desktop computer both would be aggregation.

8. INHERITANCE

Often we use words in a way that the meaning of one concept is part of another. If we say that a person travels by an automobile, the meaning does not change if he travels by a car, bus, or any other automobile vehicle. In our thought process, we use a grouping mechanism to understand and use concepts. This is a way of knowing something new in terms of something we know. What we already know has the essential features to represent the concept at that level. Without this process we could not build knowledge. When we "buy vegetables, fruits, groceries, and clothes" we have generalized some concepts.

When we say that a car is an automobile, we understand that all the structural and behavioral properties of the concept of automobile are applicable to a car. We do not have to define and understand these concepts again. This simplifies our understanding and communication. The car can have some additional property, uncommon to other automobiles. It is easier to say that a car is a special kind of automobile having all the properties of a common automobile and some stated additional features. This extends the concept of an automobile. When we define a bus, again we can use the already defined model of automobile and add features specific to a bus.
When we define a new concept in terms of already understood concepts, we inherit the features of the previously known concept and specialize it by adding some new features. As we have inherited some features of the existing model, should there be some changes to the original concepts, it would affect all the specialized concept too. E.g., if the concept of automobile assumed it to run on IC engines, all the inherited vehicles would

also have IC engines – not battery powered motors to move them. If there be a change in the concept of automobiles to change from IC engine to electrical motors, it would affect all the inherited concepts.
With inheritance we can define a new concept in terms of already known concepts, which helps build knowledge incrementally, reusing existing knowledge/model.
Contrast this with association where we use another object by knowing its responsibilities and interfaces.

9. FEEDBACK AND EXPERIENCES FROM PARTICIPANTS

We have offered this course for different types of participants with some variations in the contents. In this section, we discuss the structure of the course, feedback and reflections of various types of participants. The material in this section will help those who want to organize a course based on this paper.

9.1 Structure of the Course

It is a two and a half days course. Broadly, we cover OO Concepts, Use Case Modeling and Flow of Events during the first half day. We use the materials of this paper to explain the concepts. However, a course of this type is effective when a single case study is used to illustrate all the concepts. We use the book by Terry Quatrani (Ref [3]) for this purpose, omitting information specific to Rational Rose [it is difficult to include these features in two and half days course]. The book takes ESU online registration system as case study. We observed that the participants were able to arrive almost all the use cases and FOE related to the case study. The case study discussion lasts till the first half of the afternoon. In the remaining time of the day we explain the concept of boundary, control and entity objects. The concept of sequence diagram is explained in morning session of the second day. The first half of afternoon session is spent in understanding these concepts using the case study. The concepts of association, aggregation and composition are explained in the second half of the afternoon. The first 2 hours in the morning of third day is used to illustrate these concepts using the case study. The last 2 hours are spent in questions and answers session.

Our observation is that the participants are able to work the ESU case study more out of discussions and mutual interactions. At the end of the course they are comfortable with the book by Terry Quatrani and they can learn use of Rational Rose with 2 hours hands on session.

9.2 Reactions and Feedbacks from the participants

The participants can be divided into 3 types: (a) End User Managers (b) Entry Level Software Developers (c) Software Engineers with about 5 years experience in software industry. We give below feedback from these groups.

(a) End User Managers: There is not much change in the materials discussed. We needed to give more examples. We need to discuss in detail the generation of test transactions for each scenario of each use case. At the end of the course they are comfortable in participating in requirements gathering. Most of the participants felt that the method of delivery also helped to understand and remember the concepts. They felt that most of the course was delivered like a drama. One group of bankers remarked as follows " They are impressed with the use case way of specifying requirements. Earlier, they were capturing

requirements as documents. First time the software vendor gave the 'use case' form of requirement they did not understand it. But they confidently said that the use cases are not correct because they are not understanding the 'use cases'. Interestingly the software vendor agreed with it and revised the use case version.

(b) Entry Level Software Developers: For this group, we illustrate the realization of OO concepts by a computer. For example, object is a 'chunk of memory in heap', handle to the object is a pointer type of variable(use the 'malloc' function in C language to realize the object), sequence diagrams correspond to a high level java code, difference between database approach and object approach. The exercises would be to write use case model and FOE for amezon.com, google etc. This group of participants appreciates the manager's version of OOA when shown from the perspective of implementation.

(c) Software Engineers with more than 5 years experience: This group could grasp all the material (including requirement based testing and realization of OOA concepts using computers) in about 2 days. They felt that it is a very good exposition of OOA. Earlier, their understanding was from software perspective but viewing OOA concepts from domain people's perspective was new. They liked the concept of 'office boy as control object' and how this concept helps in the separation of concerns of boundary and entity layers. These people were also interested in the deployment view, implication of relationship between objects (association, aggregation, composition) on the design of the software system. They were interested in design patterns and related topics.

10. CONCLUSION

In the present article, we have attempted to explain OOA concepts for the managers of end user departments. We believe that these concepts are more of a rediscovery rather than new concepts to the people in the end user departments. This article can be used by the software analysts to educate the managers from end user departments about the OOA concepts. This helps to establish a common understanding between the parties involved in the requirement capturing and analyzing work.

ACKNOWLEDGMENTS

The authors acknowledge the valuable guidance and critical comments given by professor H N Mahabala and many useful suggestions given by S Nalawade.

REFERENCES

[1] Fowler, Martin. *UML Distilled*, Addison Wesley, 2003
One of the first books written on UML, it introduces UML in a gentle way with essential concepts of iterative and incremental development approach. Martin Fowler's style is excellent, making the concepts easy to understand, illustrating the context.

[2] Jacobson, Ivar. Object Oriented Software Engineering : A Usecase Driven Approach. Addison Wesley. Jacobson is the father of use case approach. He introduced the concept of use cases and described it for the first time in this book. This is a reference book with several insights to object concepts. However, terminologies and notations have since changed.

[3] Quatrani,Terry. *Visual Modeling with Rational Rose and UML.* Addison Wesley, 2002
This book will help a beginner understand and learn software development life cycle using iterative approach and UML notations.

[4] What are Objects – H N Mahabala (email: hnmahabala@iiitb.ac.in)
Private communications. An article which explains the concept and use of objects in simple language for a lay person.

[5] Booch, Grady, Rumbaugh, James and Jacobson, Ivar. *The Unified Modeling Language User Guide.* Addison Wesley, 1999
The reference manual will help clarify meaning of different standard terms in object concepts and UML. Since its publication, later versions of UML have been released.