International Journal of Engineering Sciences Paradigms and Researches, Vol. 01, Issue 01, Oct 2012
ISSN (Online): 2319-6564
www.ijesonline.com

Design of Proposed Model for Reengineering by using Object Oriented Methodology

Manjeet[1], Sandeep Dalal[2]

[1]M.tech Scholar, Department of Computer Science and Applications, MDU, Rohtak, Haryana, India
manjeetnain@yahoo.com

[2]Assistant Professor, Department of Computer Science and Applications, M.DU, Rohtak, Haryana, India
sandeepdalal.80@gmail.com

Abstract

In this paper we will represents a model that are basically used to improve the efficiency and reduce the operational cost during reengineering of software modules. While reengineering takes place on software modules, it will focus on object oriented methodology. As we know that, the public sector organizations generally follow lengthy procedures and the results are very slow. The re-engineering using object oriented methodology helps to identify the problems, which causes the delays and helps in reutilize various objects, to reduce the time. The main advantage of Object Oriented methodology is its modularity and reusability.

Keywords: *Cohesion, Coupling, Object Oriented Methodology.*

Introduction

Object Oriented Design is concerned with developing an object oriented module of software system design quality and so on.

“Object oriented design is a method of design, encompassing the process of object oriented decomposing is a notation for depicting both logical and physical as well static and dynamic models of the system under design.”

Objects are the basic units of object oriented design. Identity, States and behaviors are the main characteristics of any object. A Class is a collection of object which has common behaviors.

“A Class represents a template for several objects and describes how these objects are structured internally. Objects of the same class have the same definition both for their operation and for their information structure”

There are several essential themes in object Oriented design. These themes are mostly support object oriented design in the context of measuring and mention in below sub section**.**

Business processes are simplified rather than being made more complex. Companies that reengineer invariably end up dismantling departments and instead put together process teams that handle work logically rather than within the artificial department constraints. Inevitably, the process team approach will be more logical and make more sense than any other approach. Process teams within a reengineered organization can be of any shape or size. The work to be done dictates the optimum size and structure of the process team -- not any artificial constraints, preferences of the managers or external factor.

Reengineering will always change the boundaries between different kinds of work. In the past, the roles filled by the manager -- checking, reconciling, monitoring and tracking -- will most likely have been at the center of operations.

After reengineering, the creation of value becomes the main focus point. As such, the people who do that most effectively will become the center of focus. Teams will do whatever is required to maximize the efficiency of professionals with the skills applied.

In the reengineered business environment, advancement from one position within the company to another is not given as a reward for previous results. Instead, it's entirely ability driven.
The role and purpose of the manager change from supervisor to coach. Process teams don't need bosses - they need coaches. A boss allocates work. A coach helps the team solve problems, and facilitates achievement by providing the requisite resources and other inputs. In short, managers in reengineered companies take pride in the accomplishments of the teams they are responsible for assisting.

Internal Quality of Object Oriented Design

Cohesion

Cohesion refers to the internal consistency within the parts of the design. Cohesion is centered on data that is encapsulated within an object and on how methods interact with data to provide well-bounded behavior. A Class is cohesive when its parts are high correlated. It should be difficult to split a cohesive class. Cohesion can be used to identify the poorly designed classes.

Coupling

Coupling indicates the relationship or interdependency between modules. For Example, Object X is coupled to Object Y if and only if X sends a message to Y that means the number of collaboration between classes or the number of messages passed between objects. Coupling is a measure of interconnecting among modules in software structure.

Inheritance

Inheritance is mechanism where one object acquires characteristics from one or more other objects. Inheritance occurs in all levels of class hierarchy.

"Inheritance is sharing of attributes and operations among classes based on the hierarchical relationship."

Information Hiding

Booch states that information hiding is the process of hiding all the secrets of an object that do not contribute to its essential characteristics. An Object has a public interface and a private representation. These two elements are kept distinct. Information hiding acts a direct role in such metrics as object coupling and the degree of information hiding.

Localization

In object oriented design approach localization is based on objects. In a design, if there is some changes in the localization approach, the total plan will be violated because one function may involve several objects and one object may provide may functions.

"Localization is the process of gathering and placing things in close physical proximity to each other"

Principles of OOD

This section shows some OO design principles, which are used for support in OO design. Object Oriented Principles advise the designers what to support and what to avoid. The design principles have been categorized into three groups in the context of design metrics. These are general principles, cohesion principles and coupling principles.

General Principles

The open/closed principle (OCP) : Open Closed principle states a module should be open for extension but closed for modification. i.e. Classes should be written so that they can be extended without requiring the classes to be modified.

The Liskov Substitution principle: (LSP) Liskov Substitution Principle mention subclasses should be substitutable for their base classes i.e. a user of base class instance should still function if gen an instance of a derived class instead.

The Dependency Inversion Principle (DIP) Dependency Inversion Principle state high level classes should not depend on the low level classes i.e. abstractions should not depend upon the details. If the high level abstractions depend on the low level implementation, the dependency is inverted from what it should be.

The Interface Segregation Principle (ISP) Interface Segregation Principle state clients should not be forced to depend upon interfaces that they don't use. Many Client specific interfaces are better than one general purpose interface.

Cohesion Principles

Reuse/Release Equivalency Principles (REP) : The granule of reuse is the granule of the release. Only components that are released through a tracking system can be efficiently reused. A reusable software element cannot really be reused in practice unless it is managed by a release system of some kind of release numbers. All related classes must be released together.

Common Reuse Principle (CRP): All Classes in a Package should be reused together. If reuse one of the classes in the package, reuse them all. Classes are usually reused in groups based on collaborations between library classes.

Common closure Principle (CCP) : The classes in a package should be closed against the same kinds of changes. A change that affects a package affects all the classes in that package. The main goal of this principle is to limit the dispersion of changes among released packages. changes must affect the smallest number of released packages. Classes with a package must be cohesive. Given a particular kind of change, either all classes or no class in a component needs to be modified.

Coupling Principle

Acyclic Dependencies Principle (ADP) the dependency structure for a released component must be directed a cyclic graph. And there can be no cycles.

Stable dependencies Principle (SDP): The dependencies between components in a design should be in the direction of stability. A Component should only depend upon components that are more stable than it is .

Stable Abstractions Principle (SAP): The abstraction of package should be proportional to its stability. Packages that are maximally stable should be maximally abstract. Instable packages should be concrete.

Metrics and Quality

Since Object oriented System is becoming more pervasive, it is necessary that software engineers have quantities measurements for accessing the quality of designs at both the architectural and components level. These measures allow to designer to access the software early inn the process, making changes that will reduce complexity and improve the continuing capability of the product. The measurement process is to drive the software measures and metrics that are appropriate for the representation of software that is being measured.

Metrics are categorized into two groups

(1) Project Based Metrics
(2) Design Based Metrics

Project based metrics contain process, product and resources.

1. Process: Processes are set of software related activities which are used to measure the status and progress of the system design and to predict the future effects. A process is usually related with some timescale.

2. Product: Product metrics are used to control the quality of the software product. The metrics are applied to incomplete software products in order to measure the complexity and to predict the properties of the final product. Products are any artifacts, deliverables or documents that result from a process activity.
3. Recourse: Resources are entities required by a process activity. The resources that need to be measured include any input for software production. Thus personnel, materials, tools and methods are candidates for measurement.

Design based metrics contains traditional metrics and object oriented metrics.

Following metrics are used for Object Oriented Design to determine the complexity.

Method Hiding Factor (MHF)
Attribute Hiding Factor (AHF)
Method Inheritance Factor (MIF)
Attribute Inheritance Factor (AIF)
Polymorphism Factor (POF)
Coupling Factor (COF)

4. Method Hiding Factor (MHF): MHF is defined as the ratio of sum of the invisibilities of all methods defined in all classes to the total number of methods defined in the system under consideration. The invisibility of a method is the percentage of the total classes from which this method is not visible. In MHF, inherited methods are not considered. The number of visible methods is a measure of the class functionality. Increasing the overall functionality will then reduce MHF.
5. Attribute Hiding Factor (AHF) : AHF is defined as the ration of the sum of the invisibilities of all attributes defined in all classes to the total number of attributes defined in the system under consideration. Ideally the value of this metric would be 100%, all attributes would be hidden and only accessed by the corresponding class methods.
6. Method Inheritance Factor (MIF): MIF is defined as ratio of the sum of the inherited methods in all classes of the system under consideration to the total number of available methods for all the classes.
7. Attribute Inheritance Factor (AIF): AIF is defined as ratio of the sum of the inherited attributes in all classes of the system under consideration to the total number of available attributes for all the classes. At first sight we might be tempted to think that inheritance should be used extensively. However, the excessive re-use through inheritance make the system more difficult to understand and maintain.
8. Polymorphism Factor (POF):PF is defined as the ratio of actual number of possible different polymorphic situation for the class. In some cases overriding methods could contribute to reduce complexity and therefore to make the system more understandable and easier to maintain
9. Coupling Factor (COF):Coupling is a measure of interdependence of two objects ,for example ,objects A and B are coupled if a method of object A calls the method or access a variable in objects B. Classes are coupled when methods declared in one class use methods or attributes of other class. This factor has a very positive co-relation with all quality measures

Measuring Quality

Measurement enables to improve the software process, assist in the planning, tracking the control of a design. A good software Engineer uses measurements to asses the quality of analysis and design model, the source code, the test cases etc.

“Quality refers to the inherent or distinctive characteristics or property of object, process or the thing. Such characteristics or properties may set things apart from other things or may denote some degree of achievement or excellence.”

Impact of an object oriented approach:

Shifting of development effort into analysis: In object oriented approach much of the time is devoted t the analysis phase of the life cycle. It sometimes discouraging to spend more time during analysis and design, but this extra effort leads to faster and similar implementation. Because the resulting design is cleaner and more adaptable, it is easier to make changes in it.

Emphasis on objects before functions: This approach focuses attention n data structure instead n the functions to be performed. The concept of an object allows the development process to be more stable. Even all another concepts such as functions, relationships and events are organized around objects so that information is not lost or transformed during design and implementation phases.
Vast Development Process: Object Model during analysis is used for design and implementation and worked or the model is refined at more detailed levels rather than converting from one representation into another.
Iterative rather than sequential: The actual development process of life cycle is iterative. Each interaction adds or clarifies features rather than modifies work, which has been already done, so there are fewer chances of errors.

Procedure oriented Methodology:

This Methodology is in C language which is developed at AT&T's Bell Laboratories of USA in 1972.It was designed and written by a man named Dennis Ritchie. In the late sensitive C began to replace the more familiar languages like ALGOL, FORTRAN etc.

Steps involved in OOD

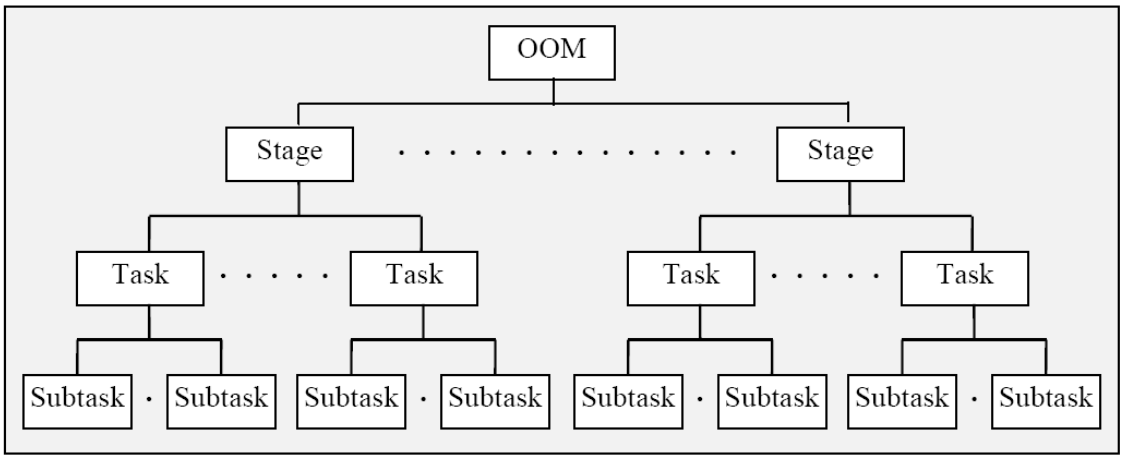
System Design: It creates the product architecture, defines the series of layers that accomplish the specific system function and identifies the classes that are encapsulated by the subsystems that are present at each layer.

Object Design: It focuses on internal details of the individual classes, defining attributes, operations and message details.

Structure of Object Oriented Methodology

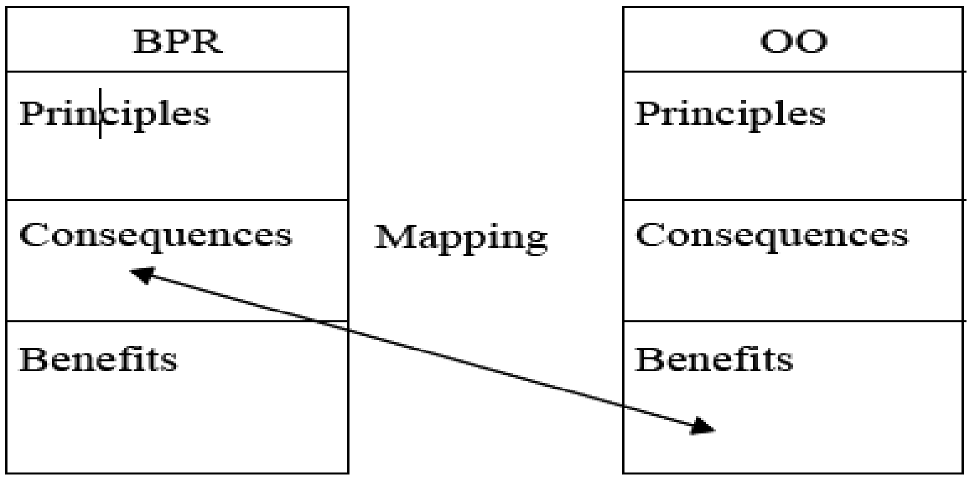
The structure of OOM is divided into Stages. Each Stage consists of a number of tasks and each task is further decomposed into sub-tasks.

The following diagram depicts clearly the structure of OOM:



Proposed Model

The model described in this paper is an attempt to delineate the central features of BPR and OO and provide a mapping of the links between them. There are two essential aspects to the model: the vertical or classification, and the horizontal or mapping.



'Proposed Model'

A review of the literature on BPR and OO identifies the essential features of each and classifies them as principles, consequences or benefits. This classification forms the vertical aspect of the model and describes a progression from the principles of the approach, through the consequences of the application of those principles, to benefits that should be realized. The idea behind this progression is that the application of a set of principles will, in turn, imply certain consequences, that ultimately should provide a set of benefits. For example, the effect of the BPR principle, "workers make decisions and the process itself has built-in controls" will be that there is "greater empowerment of individuals". Similarly, consequences should imply certain benefits. For example, the benefit that results from the OO consequence "improved means of altering components" should be "improved ability of the system to be adapted and extended".

Design of Software Module

Based on proposed model, reengineering of software module, such problem statement into class diagram and object diagram by using OO Methodology and generalize them.

Class Diagrams

Reception
Location : Receptionist/Executives
(1) Issue of OPD Cards (2) Information to Customers

Hospital Ward
Infrastructure : Number of Beds
Treatment of Patient

Operation Theater
(1) Type of Equipments (2) Availability of Operation Theater (3) Blood Bank
Operating Patients

Doctors
Qualification Experience Specialization
Consultancy Surgery

Insurance Desk
Company Name Number of Executives
Claim Approval from Insurance Company

Patient
Disease Age
Fee Treatment

Nurses
Qualification Experience
Care of Patient Assisting Doctors

Object Diagrams

Reception
Receptionist: Employee Ground Floor : Location
Skills : Customer Handling Floor : Ground Floor

Operation Theater
Blade, Cutter : Equipment Blood: Blood Bank
Metal : Steel Blade & Cutter Blood Group : B+, A+

Hospital Ward
Room : Infrastructure Iron Bed : Infrastructure
Room : AC Room Metal : Iron Bed

Doctors
MBBS : Qualification
Specialization : Cardiology

Insurance Desk
Medical Insurance : Insurance Manager : Employee
Coverage : Medical Expenses Designation : Insurance Manager

Patient
Eyes Infection : Disease
Parts of Body : Eyes

Nurses
B Sc : Qualification Designation : Senior Nurse

States:

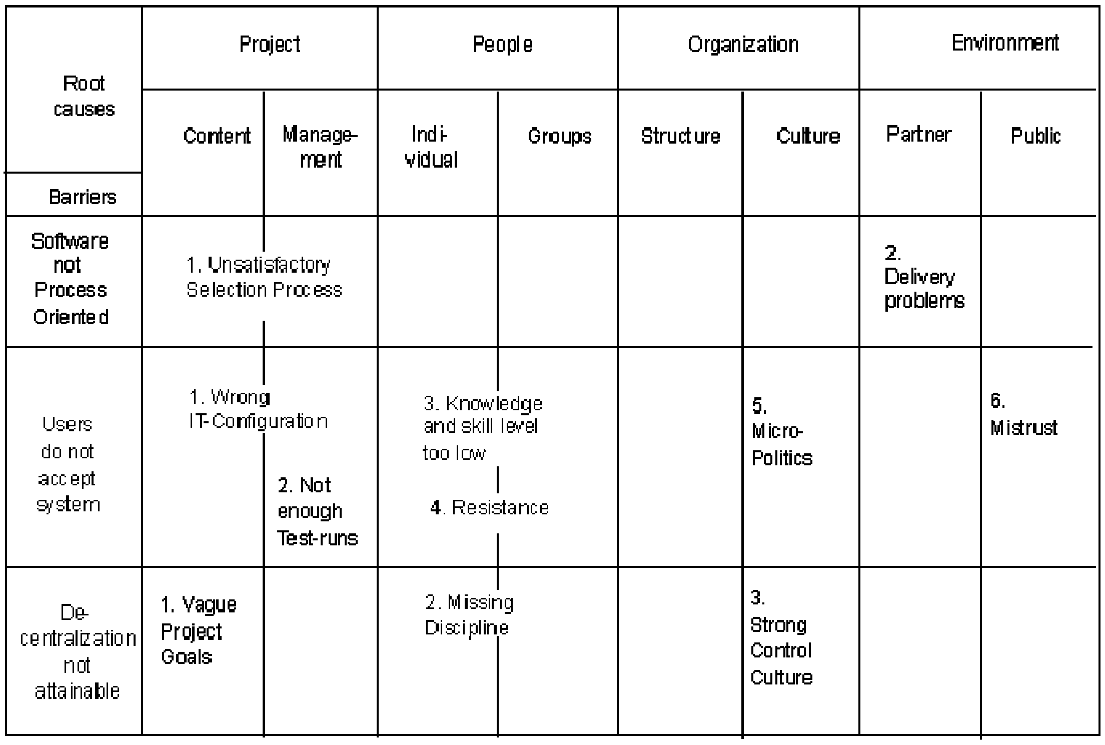
A state is an abstraction of the values and links Sets of values are grouped together into a state according to the behavior of the objects. State is represented by rounded box. State has a definite duration. It is associated with a continuous activity.
A state diagram relates events and states. A change of a state caused by an event is called transition. A state diagram describes the behavior of a class of objects. Since all objects of the same class have the same behavior they all share the same state diagram, as they all share the same class features.

Generalization

It is a relationship between a class (the super class) and one or more variations of the class (sub class). It organizes classes by their similarities and differences. The class that is being refined is called SUPER CLASS and each of its refined versions is called SUBCLASS.

Discussion

Proposed model help to set Relationships between Hard and Soft Barriers due to this we can determine individual resistance factor and information technology barrier of root cause.



Root causes	Project		People		Organization		Environment	
Barriers	Content	Manage-ment	Indi-vidual	Groups	Structure	Culture	Partner	Public
Software not Process Oriented	1. Unsatisfactory Selection Process						2. Delivery problems	
Users do not accept system	1. Wrong IT-Configuration	2. Not enough Test-runs	3. Knowledge and skill level too low 4. Resistance			5. Micro-Politics		6. Mistrust
De-centralization not attainable	1. Vague Project Goals		2. Missing Discipline			3. Strong Control Culture		

Typical Root Causes to Information Technology Barriers

Root Causes	Project		People		Organization		Environment	
Individual Resistance	Content	Manage-ment	Indi-vidual	Groups	Structure	Culture	Partner	Public
Constructive Resistance (overt and covert)		1. Approach	2. Personality			3. Former Experiences		
Destructive Resitance (overt and covert)	1. Objectives	2. Missing change momentum	3. De-structive character	4. Group Pressure	5. Loss of Power over People			6. Loss of Acceptance

Typical Root Causes of Individual Resistance

Conclusion

The hard and soft barrier helps us to understand and design of software modules, while reengineering takes place not only today but also in future.

Reference

[1] Davenport, Thomas & Short, J. (1990), "The New Industrial Engineering: Information Technology and Business Process Redesign", in: Sloan Management Review, Summer 1990, pp 11–27

[2] Davenport, Thomas (1993), Process Innovation: Reengineering work through information technology, Harvard Business School Press, Boston

[3] Davenport, Thomas (1995), Reengineering - The Fad That Forgot People, Fast Company, November 1995.

[4] Grady Booch. "Object-oriented Analysis and Design with Applications, 3rd

[5] edition":http://www.informit.com/store/product.aspx?isbn=020189551X Addison-Wesley 2007.

[6] Rebecca Wirfs-Brock, Brian Wilkerson, Lauren Wiener. Designing Object Oriented Software. Prentice Hall, 1990. [A down-to-earth introduction to the object-oriented programming and design.]

[7] A Theory of Object-Oriented Design: The building-blocks of OOD and notations for representing them (with focus on design patterns.)

[8] Martin Fowler. Analysis Patterns: Reusable Object Models. Addison-Wesley, 1997. [An introduction to object-oriented analysis with conceptual models]

[9] Bertrand Meyer. Object-oriented software construction. Prentice Hall, 1997

[10] Brett McLaughlin, Gary Pollice, David West. Head First Object-Oriented Analysis and Design. O'Reilly, 2006.