Global Journal of Computer Science and Technology
Volume 11 Issue 8 Version 1.0 May 2011
Type: Double Blind Peer Reviewed International Research Journal
Publisher: Global Journals Inc. (USA)
ISSN: 0975-4172 & Print ISSN: 0975-4350

# Design of Quality Model during Reengineering of Legacy System

By Dr Ashok Kumar, Anil Kumar
*Kurukshetra University*

*Abstract-* The purpose of this paper is design such kind of model that will improve the quality of system during reengineering[1] of legacy system that why this model is known as Quality Model. During reengineering of object oriented system[2,3], the methodology is design in such a way that will create a link/bridge between problem detection and problem correction in the legacy system, as well as simultaneously improvement in object oriented design, that can be used during later reengineering and also reduces the complexity as compared to object oriented design[4,5,6].

The further design of legacy system in such a way to specifying, how branches can be selected, how behavior is preserved and how code transformation applied. Quality of model, depends upon two factor favor and disfavor, attach to each branches, software quality is directly proportional to maintenance cost. Quality model is used for two purpose sketch and blueprint. Sketch is used a thinking tool, which help developer communicates, some aspects of a system and alternative about, what are about to be done. Blueprint intends to be comprehensive and definitive. It is used for guiding the implementation.

*Keywords:* *Reengineering, object oriented design, Quality model, Problem detection, Re-factoring.*

*GJCST Classification:* *D.2.2*


*Strictly as per the compliance and regulations of:*

# Design of Quality Model during Reengineering of Legacy System

Dr Ashok Kumar[α], Anil Kumar[Ω]

*Abstract-*The purpose of this paper is design such kind of model that will improve the quality of system during reengineering[1] of legacy system that why this model is known as Quality Model.

During reengineering of object oriented system[2,3], the methodology is design in such a way that will create a link/bridge between problem detection and problem correction in the legacy system, as well as simultaneously improvement in object oriented design, that can be used during later reengineering and also reduces the complexity as compared to object oriented design[4,5,6].

The further design of legacy system in such a way to specifying, how branches can be selected, how behavior is preserved and how code transformation applied. Quality of model, depends upon two factor favor and disfavor, attach to each branches, software quality is directly proportional to maintenance cost. Quality model is used for two purpose sketch and blueprint. Sketch is used a thinking tool, which help developer communicates, some aspects of a system and alternative about, what are about to be done. Blueprint intends to be comprehensive and definitive. It is used for guiding the implementation.

*Keywords: Reengineering, object oriented design, quality model, Problem detection, Re-factoring.*

## I. Problem Statement

There is no doubt, that OO design [4,5,6] is one of the best choice of designer, to design any software modules, however, is universal truth, that while we restructure[7] any system it is always acceptable, it will improve its efficiency, productivity, scalability and reduces the complexity as well as reduces the resources that are required during development of development of software modules.

Therefore, restructure is part of reengineering of OO design, that help transformation of a software system, without modifying its behavior that will improve its structure of the system. A common path during reengineering of OO design is to identify fragments in the subject system's design that violet principle of good design and then try to restructure the system in such a way, that minimizes these violations. Currently there is no. of approaches that can help developer and designer to identify design flows on one side and that can perform various code transformation safely on other side. While reengineering takes place on legacy system, where OO approaches are used to build a system, developer obtain a list of design flows together with their location in the system, but the necessary transformation that remove them are left to their own judgment and experience. The mapping between a specific design flow and the code transformation to remove it, together with the consequence of choosing one set of transformation over the other, is missing. Still, there is no satisfactory approach that links between the two, guiding the developer from problem detection to code transformation, that can remove the identified problem. Therefore, improvement of OO design are required.

## II. Introduction

The life of software products extends far beyond the development of first release. After being developed to the customer, a software product enter the most extended stage of its life: evolution. Software maintenance [8] is concerned with the changes, that need to be made on an existing product: defect are removed. As we know that, software evolution is a reality with various reason and consequence. As software system evolve, their structure degrades a phenomena know as software decay.

Developer who makes the changes are not same as the one who developed the initial system. For a large system it is difficult to understand the concept of initial designer had in mind, so changes are made that changes the initial concept.

Future changes are need to take into account, the new concept and execution introduced by previous changes. This leads to system very difficult to update and understand, the changes introduces more bugs and documentation became increasingly inaccurate. Maintaining these system become nightmare and the maintenance cost increase very much. Therefore, reengineering is the best choice for this purpose. During reengineering process certain phases can be identified. The one that bears restructure of OO system, where subject system is modified in order to improve its structure without affecting its functionality.

In other words, restructure process aims to improve the quality of existing system, in order to

*About[α] - Professor, Department of Computer Science & Application Kurukshetra University, Kurukshetra, India.*
*About[Ω] - Asst. Professor, Computer Science & Engg. Vaish College of Engineering, Rohtak Rohtak, Haryana (124001), India.*
*E-mail- Bestanil2005@rediffmail.com*

facilitate the later reengineering task. Restructuring is often used as a form of preventive maintenance to improve the physical state of the subject system with respect to certain standard.

The steps that will be followed by restructuring of OO system are:

- Do a survey of existing work on problem detection and re-factoring and attempts to provide a link between two.
- Select a no. of design flow that are the subject of further investigation
- Describe the way to remove such design flows from OO system
- Define methodologies for the design improvement using correction strategies


- Implement an infrastructure that allows the easy implementation of correction strategies.

The quality model is based on design pattern [9], that will improve the legacy system, during design phase, rather than code.

This model is help to determining:

- Language independence: approach should operate at design level, rather than programming language level. This way, it should be applicable to the system written in any programming language, that should support OO paradigm[10]
- Behavior preservation: any approach should give a minimum degree of confidence that the modified system will behave the same as before
- Automation: most legacy system are very large as well as very complex. Therefore, this approach should allow for large degree of automation and should minimizes human intervention during the process
- Quality estimation
- Extensibility easy to accommodate, new knowledge
- Causality: this approach should directly detect any risk at design flows.

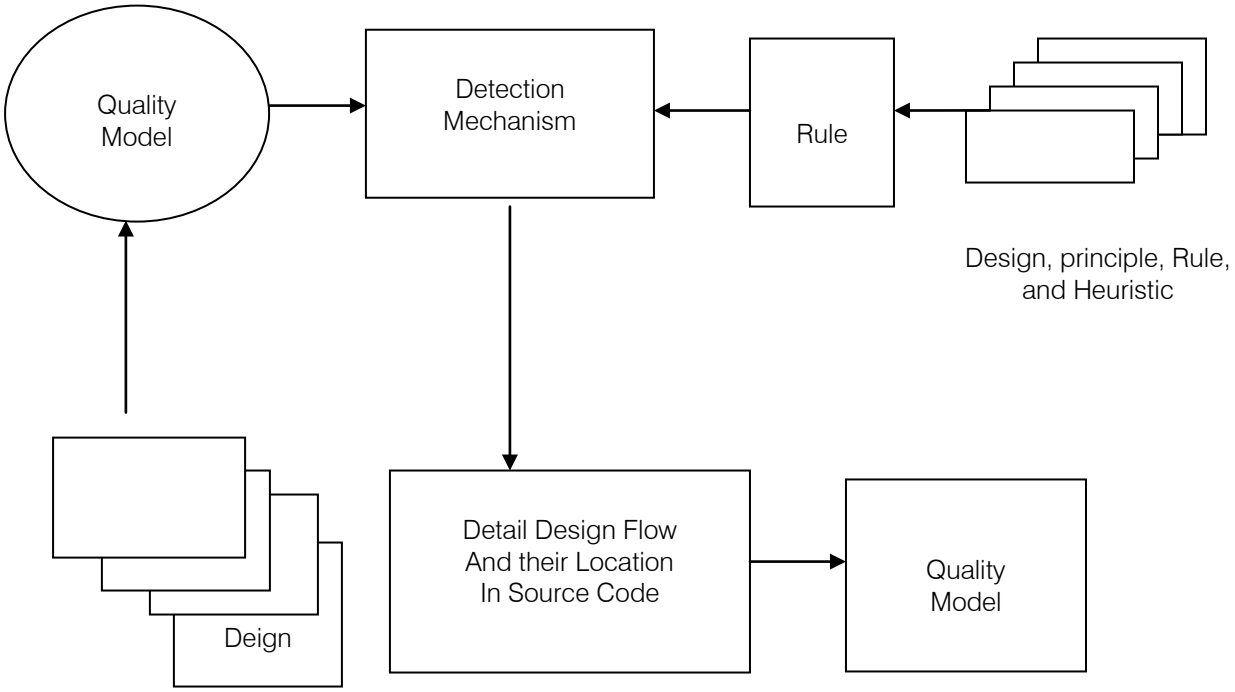
Reengineering life cycle [11,12,13] contain problem detection phase, when design flows are modified.

Each pattern describe a problem, which occurs over and over again in one environment and the describe the core of the solution to that problem, in such a way that you can use this solution a million of times over, without ever doing it, the same way twice.

Design pattern, is the description of communicating objects and classes, that are customized to solve a general design problem in a particular context. Here, we do not inspect all area of reengineering, but focus on two phases that aim to link: problem detection and re-factoring.

## III. Problem Detection

Problem detection is a specific phase in the reengineering lifecycle. Its process aims to identify design flows in the analyzed system. It has a series of steps – first formalization, which takes OO design heuristics, rules, principle and turns them into precise rules and that can be used to identify design flows. The source code is parsed and a model of the system is obtained.




*"Problem Detection Model During Reengineering of OO System"*

It will support the assessment and improvement of the quality in OO system . the problem detection strategies on the model's entities that is packages, classes, methods, attributes, local variables, global variables and parameters.

## Re-Factoring

Re-factoring[14] is the process of changing a software system, in such a way that it does not alter its external behaviors, but still improves its internal structure. The framework are reusable architecture , they are the result of many iterations and no. of way limited to OO framework [15,16,17,18] and more recent software development process, suggest, re-factoring should be part of development cycle.

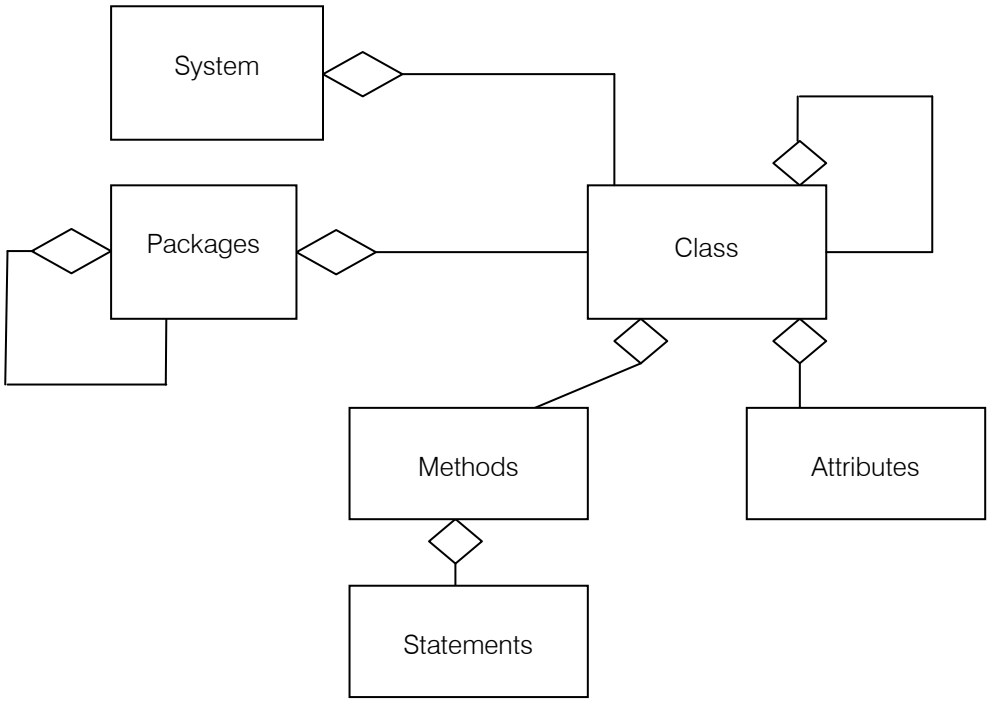
Re-factoring OO System [10]:

- Defining an abstract super class of one or more existing classes
- Specializing a class by defining sub class and using sub-classing to estimate conditionals
- Changing how the whole/part relationship is modeled ( from inheritance to aggregation)
- Moving a class within or among inheritance hierarchies
- Moving member variable and function
- Replacing a code segments with a function call
- Changing name of classes, variables or functions
- Replacing unrestricted access to member variable with a more abstract interface

Re-Factoring are further classified as:

- Low level re-factoring, such as rename, move, replace, a code segment with a function call
- High level re-factoring, such as defining as abstract super class, replacing conditionals with polymorphism and changing inheritance to aggregation. These re-factoring make use of other low-level re-factoring

Re-factoring have to guarantee, that they preserve the behaviors of the system.

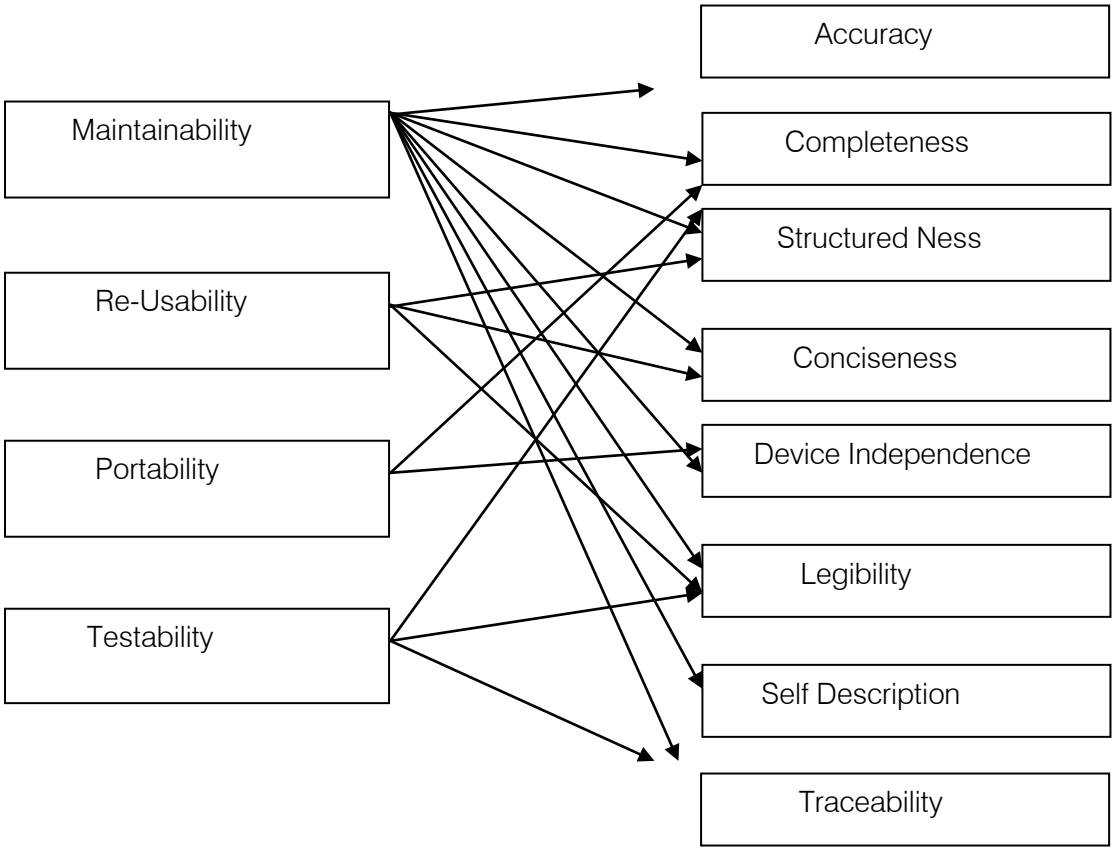



*"Pattern Based Quality Model"*

The further design of legacy system, in such a way to specifying, how branches can be selected, how behaviors is preserved and how code transformation applied. Here, problem detection model, are used to choose the appropriate branches in each decision node and safely applying code transformation. By applying this, finding a path through graph strategies. Depending on the system and reengineering goals, different path can be chosen for the same design flow.

Reengineering process, start with requirement analysis, which establish the overall reengineering goals. Reengineering goal provides the focus of all activities carried out, during the process of reengineering. These goal can be expressed in terms of quality factors, that need improvement: flexibility, portability, efficiency etc..

At each decision node, the available branches can be selected according to their impact on each of the considered quality factors.




## References Références Referencias


1. Muthu, S., Whitman, L. and Cheraghi, H. S., 1999. "Business process reengineering: a consolidated methodology", Proceedings of the 4th Annual International Conference on Industrial Engineering Theory, Applications and Practice. Retrieved October 19, 2007,
2. Grady Booch. "Object-oriented Analysis and Design with Applications, 3rd edition":http://www.informit.com/store/product.aspx?isbn=020189551X Addison-Wesley 2007.
3. Rebecca Wirfs-Brock, Brian Wilkerson, Lauren Wiener. Designing Object Oriented Software. Prentice Hall, 1990
4. A Theory of Object-Oriented Design: The building-blocks of OOD and notations for representing them (with focus on design patterns.)
5. Martin Fowler. Analysis Patterns: Reusable Object Models. Addison-Wesley, 1997. [An introduction to object-oriented analysis with conceptual models]
6. Bertrand Meyer. Object-oriented software construction. Prentice Hall, 1997.
7. Paolo Tonella. Concept Analysis for Module Restructuring. *IEEE Transactions on Software Engineering*, 27(4):351–363, April 2001.
8. Norman Wilde and Ross Huitt. Maintenance Support for Object-Oriented Programs. *IEEE Transactions on Software Engineering*, SE-18(12):1038–1044, December 1992.
9. Paolo Tonella and Giuliano Antoniol. Object Oriented Design Pattern Inference. In *Proceedings of ICSM '99 (International Conference on Software Maintenance)*, pages 230–238. IEEE Computer Society Press, October 1999.
10. Object Modeling and Design Strategies by Sanjiv Gossain, Ian S. Graham - SIGS Books & Multimedia; ISBN: 052164822X
11. Davenport, Thomas & Short, J. (1990), The New Industrial Engineering: Information Technology and Business Process Redesign, in: Sloan Management Review, Summer 1990, pp 11–27
12. Davenport, Thomas (1993), Process Innovation: Reengineering work through information technology, Harvard Business School Press, Boston

13. Davenport, Thomas (1995), Reengineering - The Fad That Forgot People, Fast Company, November 1995.
14. Martin Fowler: *Refactoring: Improving the Design of Existing Code*, Addison Wesley, 1999
15. Bosch J,Molin P,Mattson M, Bengtsson P. Object oriented frameworks—problems and experiences. *Building Application Frameworks*, Fayad ME, Schmidt DC, Johnson RE (eds.). Wiley & Sons, 1999.
16. Bosch J. Design of an object-oriented framework for measurement systems. *Object-Oriented Application Frameworks*, Fayad ME, Schmidt DC, Johnson RE (eds.). Wiley & Sons, 1999.
17. Mattsson M, Bosch J. Framework composition problems, causes and solutions. *Proceedings Technology of Object-Oriented Languages and Systems*, U.S.A., August 1997.
18. Mattsson M. Object-oriented frameworks survey of methodological issues. *Licentiate Thesis*, Department of Computer Science, Lund University, 1996.