ISSN No. 0976-5697

Volume 2, No. 6, Nov-Dec 2011

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

Decision Making Models for Reengineering of Object Oriented Software Systems

Bakhshish Singh Gill*
Sr. Programmer
Computer Centre, G.N.D. University
Amritsar, India
bbssgill@yahoo.com

Dr. Ashok Kumar,
Professor
Dept. Computer Science and Applications
Kurukshetra University,
Kurukshetra

Abstract: Reengineering of object-oriented software is not much grown area. It is difficult for the software managers when to decide for Reengineering software system. Software is delivered, it is corrected, and maintenance starts from the very beginning. Maintenance cost increases with time as requirements, technology and environment changes. The accumulated effects of maintenance make the system complex and quality of software decreases. As a result, maintenance cost increases rapidly. A situation comes when maintenance cost is too much high and it is difficult to maintain the system at such a high cost. This is a good time for reengineering the software system. If we do not reengineer the software system at this optimal time and go for high maintenance cost then complexity increases and system quality is worsen. It will be difficult to reengineer the system or reengineering cost is not justified as compared to the cost of new system. Software managers have no options except for having costly new software. Little efforts are done in this piece of work to escape this situation.
Models of Decisive Point (right time for reengineering) are presented and it will help the software managers to reengineer the software system at most favorable time.

Keywords: Object, reengineering zone, maintenance zone, decisive point, fine object, faulty object

I. INTRODUCTION

The ability to accurately estimate the time and cost of reengineering software is the key factor for successful of reengineering project. And more important is the right time decision for reengineering software system. Right time decision affects not only the cost of reengineering but also possibility for reengineering. If we cross this crucial point, reengineering is costly.

We can come across at a stage when there is no option other than purchasing new software. But when is the right time? What is that crucial point? It is that stage when software is best fit to reengineering. As we go past this stage, reengineering will be costly. Accumulated affects of maintenance makes the system complex and deteriorate the system's architecture. Software system goes on aging with time and maintenance cost increases. When maintenance cost is too much high or difficult to maintain, it means system is to retire. Then reengineering is solution at this point. With reengineering software starts working and has another life span. Reengineering should be done at right time. If we overlook this occasion, reengineering will be costly or not possible and then we have to throw the costly legacy software underutilized.

But when is the right time? How it is to be determined? These issues are addressed in this piece of work. This is very useful for Software managers and they will be aware of this point after going through this piece of work.

II. SIGNIFICANCE OF DECISIVE POINT

'*Decisive point*' is the new term coined in the field of software reengineering. It is the best fit time for reengineering software system. If we go past this point (stage) reengineering will be costly and even difficult. Software managers must focus on this point for reengineering the software to increase the life span of software. The best fit time for reengineering is called the *Decisive point*. At this time reengineering cost will be approximately 25% of the cost of new system. We can otherwise say that if cost of reengineering is 25% of the cost of new system then it is the most suitable time for reengineering. According to research paper "Cost Model for Reengineering an Object Oriented Software System' by the same authors is 25% of the cost of the new system.

Software maintenance starts after delivery of the software to correct faults, to improve performance and other attributes of the software. Maintenance plays an important role in the life cycle of a software system. Maintenance is the last stage of the software life cycle. After the product has been released, the maintenance phase keeps the software up to date with environment changes and user requirements changes. With recurring maintenance, complexity increases and software quality decreases. As the software is maintained, errors are introduced. Many studies have shown that each time an attempt is made to decrease the failure rate of a system, the failure rate got worse. On average, more than one error is introduced for every patch up error. In this way maintenance cost goes on increasing with time. Software maintenance can account for 60 to 80 percent of the total life cycle of software product. More than 90 % of the total cost of software goes to maintenance and evolution of the software product [1].

After a certain period, a stage comes when it is difficult to maintain the system or maintenance cost is too much high. Maintenance problems are a driving force behind re-engineering. Reengineering is the only way to avoid new development cost. But what is the right time for reengineering software? If software managers do not know the right time, how can they go for reengineering?

III. MODELS FOR DECISIVE POINT

I present the following three models

A. *Thoroughfare decisive point:*

After the software product is delivered, the maintenance phase keeps the software up to date with environment changes and changing user requirements. Maintenance starts at the beginning (when the software is delivered) at the point A in figure 1. As system ages, maintenance cost goes on increasing. Maintenance cost is maximum at the point D in the figure 1. In this model, life span of software is divided into two zones.

a. Maintenance zone.
b. Reengineering zone.

Software system is candidate for maintenance in the first zone from the point A to D. Maintenance cost at the first end (point A) is lowest and highest at point D with red color which is the end of maintenance zone. Beyond this point maintenance cost is not justified. We can maintain the software in the reengineering zone but with unjustified cost. Reengineering cost is lowest at the point D and highest at the point B (end point of the Reengineering Zone). After this point, reengineering can be with unjustified cost.

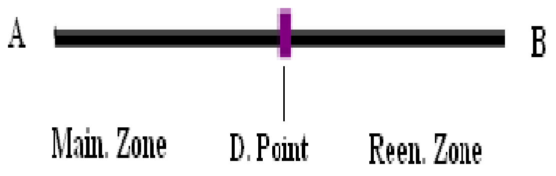

Figure. 1

As in figure 1 above, red point D in the middle is vital point, best fit time for reengineering; we call it *Decisive point*, a new term coined and added to the field of reengineering. In the maintenance zone, reengineering is not feasible because maintenance cost is ordinary. Reengineering cost and maintenance cost are equal at point D. After this point on the line maintenance cost increases rapidly and reengineering cost increases slowly as it is reengineering zone. Managers must reengineer the system at the red point to have another life span of the software with low (Normal) maintenance cost. If software managers do not reengineer the system and maintenance zone goes overlapping the reengineering zone with high maintenance cost, they will struck at a situation when maintenance is not possible and reengineering cost is also not rational and there is no alternative except to retire the software system. Reengineering cost will be optimum at the red point therefore software managers must stick to this point for financial benefits on the software.

B. *Decisive Point based on:*

Maintenance/reengineering cost

Following figure Fig. 2 depicts the graph of maintenance cost and reengineering cost of Software system. Reengineering cost starts from the point D and maintenance cost starts from the point O (Origin). Reengineering cost and maintenance cost at point D are equal. If both the costs are equal then we must go for reengineer. Reengineering will make the system new on the new platform with new design. After this maintenance cost increases with high rate and reengineering of the system is needed to bring down the maintenance cost. At this point we think of reengineering or retiring the software. If we retire the system then we have to bear the cost of new software. Cost of new software is much high than the cost of reengineering.

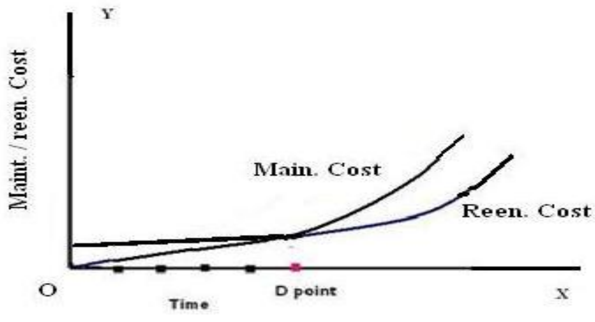

Figure. 2

If we do not reengineering the software system at point D maintenance cost will increase sharply (as shown in the figure 2), it will be difficult to maintain the system at such a high cost. Maintenance after the point D increases the complexity of the system and decreases the quality of software where as reengineering improves the quality of the software, controls the maintenance cost and increases the life span of the software system.

Why maintenance cost is high beyond red point? The age of the software is near about 7 years for the structured software systems where as the age for object oriented software system is 10 years. And at the red point the age of the software is 10 years and software is to retire or reengineer. The software system is old at this point and high maintenance cost is required. It is difficult to maintain the system with such a high Maintenance cost. At this point system should be reengineered or retired. If we reengineer the software at this point, Reengineering cost will be lowest (optimal). The point D in the fig. 2 is significant, the cost of Reengineering and Maintenance are equal so we must go for reengineering. Reengineered Software will be new one with another life span and Maintenance cost will be ordinary.

C. *Decisive Point based on faulty objects:*

This is object based model for decision making about reengineering of the software system. It is to be determined on the basis of the faulty objects. In this work, object is seen at a higher level of abstraction and is taken as conceptual module that can be plugged in and plugged out from the software system. Reengineering identifies reusable components (objects) and analyzes the changes that would be needed to regenerate them for reuse within new software architecture. The use of a repeatable, clearly defined and well understood software objects, has make reengineering more effective and reduced the cost of reengineering. Decisive Point will also be determined on the basis of objects.

a. *What is an Object?*

I thought of an object in the context of object-oriented technology as independent component that can be pulled out and put in the software system. If we pull out an object from a software system, it is working system without much affecting the whole system except the job done by that particular object. As in the other physical systems, a component is plugged out, repaired and plugged in the system again. Additional screws, nuts and bolts are required for this purpose. We must develop a universal language of such type. We must have a set of additional instructions as nuts and bolts to plug-out and plug-in the objects in the system. An object with data and instructions is depicted in the following figure 3.

Figure. 3 An Object

Data and operations are bundled and kept private from rest of the software as in Fig. 3

The object oriented approach attempts to manage the complexity inherent in the real world problems by abstracting out knowledge and encapsulating it [2]. Object is an instance of a class and has an identity and stores attribute values [3].

Here in this piece of work, *Object* is seen at a higher level of abstraction and is taken as independent module or unit that can be plugged in or plugged out of the software system. All objects of the candidate software system are untied (***Reverse engineering***). Faulty objects are indentified and modified. Then redesigning of the structure (***transformation of the architecture***) of the system according to new modern design is done. Then according to new design objects are integrated (***Forward Engineering***).

b. Object-Oriented software system:

Following is the example of object oriented software system with eight objects like real world objects. Circles are objects and lines represent communications to send messages between objects. The object in the system is characterized with three properties Identity, State and Behavior. Identity distinguishes it from others, state is the data stored in it and behavior describes the methods by which the objects can be used.

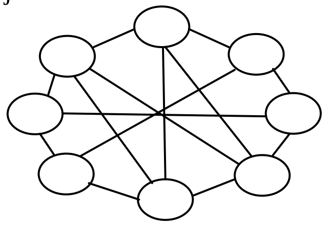

Figure. 4 8 objects software system

Abstraction is good tool for reengineering object oriented design as it helps in reducing complexity. Large systems are complex having more objects as each additional object increases the complexity of the system [4].

Object-oriented paradigm has changed the scene for reengineering. In object oriented paradigm data and procedures are combined. In object oriented approach the role of UML is supreme. It was designed to provide a standard for software modeling languages. It is a graphical notation for object-oriented analysis and design. UML provides a framework for describing a set of models that capture the functional and structural semantics of any complex information system. An object is small piece of source code that can be reengineered independently. Object-oriented software system is all about objects. Object-Oriented software system is being more reusable and hence more suitable for reengineering. Reengineering of software system is accomplished by reengineering the faulty objects in the system. Software system is untied, objects are identified for reengineering. Identified objects for reengineering are called faulty objects. Faulty objects are reengineered independently and made Fine objects, software architecture is changed, and all the objects (now all objects are fine) are integrated according to the new architecture.

c. Reengineering Decision Model:

Let our candidate system be an object oriented system with N fine objects. Fine object is an object which conforms to our requirements and functions well in the system. As software ages some objects becomes faulty. Faulty object is an object which does not conform to our requirements and does not function well with in the system. We go on maintaining the faulty objects to maintain the software system. With maintenance of the faulty objects again and again, architecture of the software deteriorates. Maintenance cost also goes on increasing. We reach at a point where reengineering of the system is needed. These faulty objects can be reengineered and can be plugged again to get the system reengineered. Let us suppose there are N objects in system which is our candidate system. Let it be O1, O_2, O_3,……………..O_N. As the system ages the faulty objects goes on increasing. With maintenance the system gets evolved, correction of one error gives birth to another. Go on maintaining the software till half of the objects are not faulty. When half of the objects (N/2) are faulty in your application go for reengineering the software. The reengineering cost of the candidate system with N/2 faulty objects will be one forth (25%) of the new development cost [5]. This is the optimal cost according to the research paper ‘Cost of Reengineering (Object-Oriented Software Systems) versus Developing new One- A Comparison’ by the same author. Hence you reach the *Decisive point* when N/2 objects are faulty in the software with N objects.

Decisive point (the right time) is when N/2 or more objects are faulty (System with N objects). When N/2 objects become faulty; Reengineering Zone starts. This is vital time in the software life cycle for reengineering. If the software managers pay no heed to this time they have to retire the legacy software and go for new one. Reengineering will not be feasible after this crucial time. It will be financial loss to the organization as the recourses are underutilized.

Software managers should not ignore Decisive Point otherwise they have to retire the underutilized software system.

IV. RESULTS AND CONCLUSIONS

In this piece of work three decision making models are presented as under

a. Thoroughfare decisive point
b. Decisive Point based on Maint. / Reeng. Cost
c. Decisive Point based on objects

These models are valuable to software managers for reengineering the software systems at the right time. The right time is red point on the life span of software. Reengineering is not feasible before and after the red point. These models will help to reengineering the software and escape the burden of purchasing costly new software. Software investment expenditure curve will fall in the organizations. There will be full utilization of the software and software backlog will be decreased.

In this work three new terms 'Decisive point', 'Reengineering Zone' and 'Maintenance Zone' are coined and added to reengineering subject matter.

V. FUTURE WORK TO BE DONE

These given Models are new in the field of Reengineering. The future work is to test these models for suitability to fit on the basis of analysis of current and past data. Near about 50-80 projects can be judged to fit these models. These models are to be tested and accepted or improved or rejected. Once fit and fine these models will help in reengineering the legacy software with optimal cost.

This work will be beneficial to the both communities, the software managers and the software engineers. Software managers will save the software expenditure and engineers will get the much work on software reengineering.

VI. REFERENCES

[1]. Erlikh, L., "Leveraging legacy system dollars for E-business". *(IEEE) IT Pro*, May/June 2000, PP. 17-23. Down loaded on 24-02-2011 from the site: http://users.jyu.fi/~koskinen/smcosts.htm

[2]. Brock R.W.,Wilkerson B., Wiener L. , "Designing Object-Oriented Software", 2007, Prentice-Hall of India, New Delhi pp. 5

[3]. Bernd Bruegge, Dutoit Allen H. , "Object-Oriented Software Engineering Using UML, Patterns, and Java", 2004, Pearson Education (Singapore), pp.724

[4]. Halladay S., Wiebel M., "Object Oriented Software Engineering", BPB Publications, New Delhi. pp. 35

[5]. Bakhshish Singh Gill, 'Cost of Reengineering (Object-Oriented Software Systems) versus Developing new One- A Comparison' Research paper, Serials Publication, New Delhi, 1-04-2011