Global Journals LaTeX JournalKaleidoscope™
Artificial Intelligence formulated this projection for compatibility purposes from the original article published at Global Journals. However, this technology is currently in beta. *Therefore, kindly ignore odd layouts, missed formulae, text, tables, or figures.*

Maintenance vs. Reengineering Software Systems

Bakhshsish Singh Gill[1]

[1] Guru Nanak Dev University

Received: 7 November 2011 Accepted: 5 December 2011 Published: 20 December 2011

Abstract

Maintenance and reengineering terms are closely coupled with each other. These terms came from the world of hardware objects. Now these entered the world of software and are well suitable for software systems. It is difficult to draw a clear cut line between these two terms. Many a times these are used interchangeably. Reengineering of software systems is a topic of importance and in coming time it will be gaining more attention in the world of software systems. Software managers are often confused over maintenance and reengineering. These two terms should be separated to promote the subject matter because one is problem for the other. I will try to put them in different non overlapping regions. Maintenance and reengineering are two different areas in software engineering. Maintenance is for running the system till the age of the system where as the reengineering make the system new to work for another life span. Scope of reengineering is vast and challenging as compared to maintenance. Reengineering is to reduce the expenses on software systems in the organizations. Reengineering has more scope in the world of software than in the world of hard ware objects. Software systems and software objects do not wear and tear out like hardware objects in the real world. Maintenance is close to repair/mend where as reengineering is very close to new development.

Index terms— Object, reengineering zone, maintenance zone, transition state, reverse engineering

1 INTRODUCTION

oftware engineering is a topic of importance in the age of software and is gaining attention. It is developing fast area and not existing from centuries. Software maintenance and software reengineering both fall in the ambit of software engineering. Both terms came from the real hard ware objects. These are more suited to software systems and software objects as these do not wear or tear out like real world physical objects.

These two terms are yet young and developing. There is not clear cut line between them. These terms are mingled and the people are using them interchangeably. One is the problem in the developing the subject matter of the other. Now software is gaining importance in every sphere of life and these two are very closely associated to software system life cycle. It is time to differentiate the two and promote the subject matter of these two concepts.

2 II.

3 SOFTWARE MAINTENANCE

Software maintenance is one of the stages in the software development life cycle. It starts after the deployment of software in the working field. It is to remove the defects and deficiencies which encounters while starts actually working in the field.

According to IEEE Std. 610.12 **???**] 'Software maintenance is the process of modifying a software system or component after delivery to correct faults, improve performances or other attributes, or adapt to a changed

environment'. a) Nature and Scope of Maintenance Software maintenance has good scope in future times. In the world of fast changes, maintenance expertise will gain more importance which can sustain the working of software systems. Maintenance means modifying software system or a component of the software system to make it working on the platform and adapt to the minor changes in the requirements, environment or technology. Every system needs maintenance for the whole life period. Maintenance is preservation of the legacy software system.

Many surveys have shown that software maintenance can account for 60 to 80 percent of the cost of total life cycle of software product. According to Erlikh more than 90 % of the total cost of software goes to maintenance and evolution of the software product **??**1]. According to Lientz and Swanson many organizations were spending 20% to 70% of their computing efforts on maintenance **??**8].

Software maintenance is of following four types

4 Corrective maintenance:

As its name implies, activities of correcting bugs in the software are included in this type of maintenance. It is for making the software system to conform to the real situation.

5 Adaptive Maintenance

It deals with making the software adjustable to the changed environment

6 Preventive Maintenance

Modification of the software to detect and correct hidden faults (bugs) before becoming active. This paper will help the software managers to recognize and make the best use of these two terminologies for right treatment of software systems.

7 Perfective maintenance

It is modification of the software for better performance, maintenance and reliability. The activities related to updating the software are included in this type of maintenance.

The efforts (cost) distributions for these four types of maintenance are as under? Corrective maintenance 21% ? Adaptive Maintenance 25% ? Preventive Maintenance 4% ? Perfective maintenance 50% [6].

It is suggested from the above figure that perfective maintenance consumes major part of cost estimation. Preventive maintenance is not taken to any significant level in software industry.

Software maintenance and reengineering are hot topics in theses days. Software managers use these two interchangeably. It is time to differentiate maintenance and reengineering in software industry. Software maintenance is last stage in the software development life cycle. Maintenance starts after the delivery of the software. The ability to accurately estimate the time and cost of software maintenance is the key factor for successful of maintenance project.

Software maintenance starts after delivery of the software system. It goes on increasing with the increasing age of software as depicted in the following figure **??**.

Figure **??** 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. Reengineered software starts working with normal maintenance for another life span. Reengineering should be done at right time. If we overlook this time, reengineering will be costly or even not feasible and then we have to throw the costly legacy software under utilized.

III.

8 REENGINEERING

Reengineering is the analysis of existing software system and modifying it to constitute into a new form. Chikofsky and Cross define reengineering as 'the examination and alteration of a subject system to reconstitute it in a new form and subsequent implementation of that form' [9].

According to IEEE Std. 1998 'A systemchanging activity that results in creating a new system that either retains or does not retain the individuality of the initial system' [10]. a) Nature and Scope of Reengineering When maintenance cost is not feasible, we go for reengineering the software system. Reengineering makes the software system new. Reengineering has the following three stages.

9 Reverse engineering 2. Architecture transformations 3. Forward engineering

10 Reverse engineering

In this stage software is thoroughly understood. It is untied and underlying technology is perceived. Business process is improved and requirements are updated. Objects are added or deleted according to the new system planned. In this stage we go from code level to higher level abstraction.

11 Forward engineering

In this stage, we move from higher level of abstraction to code level. In this stage software integrated according to new design. It is vertically downward step as shown in the fig. 2.

Figure 2 In the above figure, updation of user's requirements and improvement in architecture of software is done in transformation phase.

We can express reengineering by the following equation.

12 Reengineering = Reverse engineering + ? + forward engineering

The symbol in the above equation represents the enhancement and design change. In the coming future, reengineering can solve the problem of software backlogs and it can lower the software investment in organizations. Reengineering can increase the software age. Point T is the transition state from maintenance to reengineering zone. Transition state is new term defined by the author of this paper. It is the state in the life of software when reengineering is best possible with optimal cost. Software system is candidate for maintenance in the first zone and candidate for reengineering in the second zone. Maintenance phase is always first and reengineering phase starts after maintenance phase. When maintenance exhausts, reengineering phase is ready to serve the software system. Both zones are separated by red point T and must not overlap each other. Figure **??** Maintenance increases the age of the software and reengineering gave a fresh age period to software system. T is the transition point, beyond point T; it is not feasible to maintain the system. System should be reengineered at the point T. Reengineering cost will be optimal at the critical point T. If we do not reengineering the System at T and go on maintaining the software with high cost, reengineering zone will be exhausted and reengineering is not possible with feasible cost. Then there is no other option than to throw the legacy software and purchase costly new one. Legacy software will be added to the backlog of wasted software.

Maintenance phase keeps the software up to date with environment changes and changing user requirements. Reengineering will give another life span to software with normal maintenance.

13 b) Cost based Model

Following figure **??** depicts the graph of maintenance cost and reengineering cost of Software system. Maintenance cost starts from the point O (Origin) and goes on increasing with time. It starts increasing rapidly from point T because software completes ten years, the normal age of the software. According to literature, software age is seven years for structured systems and ten years for object-oriented software systems. Reengineering cost is all most same up to point T because software is within age at the point T. After point T reengineering cost also starts increasing but with normal rate but maintenance cost increases at high rate. This happens because maintenance zone is over and the software is in reengineering zone beyond point T. Maintenance cost and reengineering cost are equal at the point T as depicted in the figure **??**. 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. 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 **??** If we do not reengineering the software system at point T, maintenance cost will increase sharply (as shown in the figure **??**) it will be difficult to maintain the system at such a high cost. Maintenance after the point T 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.

The software system is old at the point T 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). Reengineered Software will be new one with another life span and Maintenance cost will be ordinary.

14 c) Object based Model

This is object based model for differentiation of maintenance and reengineering. Maintenance is done to make the faulty object fine. As the system ages, software architecture deteriorate with ripple effects of maintenance.

System object becomes faulty and maintenance makes it fine. The number of faulty objects increases with time and maintenance becomes difficult. Then what to Do? Software should be reengineered but when? This is the question. 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. Maintenance and reengineering will be separated on the basis of faulty objects.

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].

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).

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]. 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.

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. We reach at a point where reengineering of the system is needed. But what is that point? Let us suppose there are N objects in system which is our candidate system. Let it be O1, O 2 , O 3 ,?????..O N .

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 stage where reengineering starts.

When N/2 or more objects are faulty (System with N objects) stop maintenance and reengineer software system. When N/2 objects become faulty; it is a transitional state from maintenance to reengineering. This is vital stage in the software life span for transition from maintenance to reengineering. If the software managers pay no heed to this transitional stage and go V.

15 SUMMARY AND CONCLUSIONS

In this piece of work four models are presented for differentiation in maintenance and reengineering as under These models are valuable to software managers for reengineering the software systems at the right time. Reengineering is not feasible before and after the transition state. 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. [1] [2] [3] [4] [5] [6] [7]

[1] © 2011 Global Journals Inc. (US) Global Journal of Computer Science and Technology Volume XI Issue XXIII Version I 59 2011 December

[2] © 2011 Global Journals Inc. (US) Global Journal of Computer Science and Technology Volume XI Issue XXIII Version I 60 2011 December Maintenance vs. Reengineering Software Systems

[3] © 2011 Global Journals Inc. (US)

[4] DecemberMaintenance vs. Reengineering Software Systems

[5] © 2011 Global Journals Inc. (US) Global Journal of Computer Science and Technology Volume XI Issue XXIII Version I 63 2011 December Maintenance vs. Reengineering Software Systems

[6] © 2011 Global Journals Inc. (US) Global Journal of Computer Science and Technology Volume XI Issue XXIII Version I

[7] DecemberMaintenance vs. Reengineering Software Systems

Figure 1:

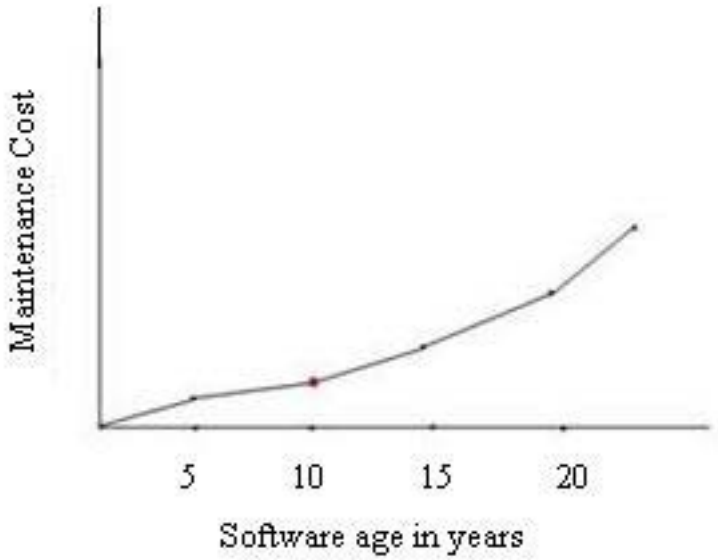
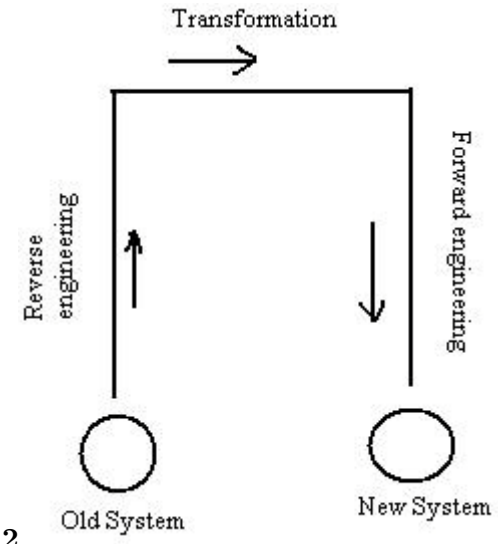


Figure 2:



2

Figure 3: 1. Thoroughfare differentiation model 2 .

1

Maintenance vs. Reengineering Software Systems
2011
December
62
on maintaining with high cost, it means they are overlapping the reengineering zone. It this way, they will strike in a situation when reengineering is not feasible. The cost of reengineering is very much high or equal to the new development. Then they will have to retire the legacy software. It will be financial loss to the organization as more investment on software is needed to purchase new software.

[Note: © 2011 Global Journals Inc. (US)]

Figure 4: Table 1

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

.1 VI.

.2 FUTURE WORKS TO BE DONE

These given Models are new in the field of Reengineering of object oriented software systems. The future work is to test these models for suitability to fit on the basis of analysis of current and past data. These models can be accepted as it is 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 can order for reengineering at transitional point where maintenance zone ends and reengineering zone starts.

[Jalote ()] *An Integrated Approach to Software Engineering*, P Jalote . 1996. New Delhi: Narosa Publishing House.

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

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

[Brock et al. ()] *Designing Object-Oriented Software*, R W Brock , B Wilkerson , L Wiener . 2007. New Delhi: Prentice-Hall of India. p. 5.

[Gill Nasib and Singh ()] Gill Nasib , Singh . *Software Engineering: software reliability, Testing and Quality Assurance*, (New Delhi) 2002. Khanna Book Publishing Co.(P) Ltd.

[IEEE Standards Software Engineering ()] IEEE Std 1219-1998. *IEEE Standards Software Engineering*, 1999. IEEE Press. Two. (Process Standards)

[Erlikh (200)] *Leveraging legacy system dollars for Ebusiness*, L Erlikh . `http://users.jyu.fi/~koskinen/smcosts.htm` 200. p. . (IEEE) IT Pro. Retrieved 24-02-2011 from)

[Aggarwal and Singh ()] *New age International*, K K Aggarwal , Yogesh Singh . 2002. New Delhi: P) Ltd., Publishers. (Software engineering)

[Booch ()] *Object Oriented Analysis and Design with Applications*, Grady Booch . 2003. Singapore: Pearson Education.

[Halladay and Wiebel] *Object-Oriented Software Engineering*, S Halladay , M Wiebel . New Delhi: BPB Publications. p. 35.

[Chikofsky and Cross (1990)] 'Reverse Engineering and Design Recovery: A Taxonomy'. E Chikofsky , J H Cross . *IEEE Software Engineering journal* Jan. 1990. p. .

[Pressman ()] *Software engineering*, Roger S Pressman . 1992. New York: McGraw-Hill. (3 rd ed.)

[Ian Sommerville ()] *Software Engineering*, Ian Sommerville . 1994. Singapore: Addison-Wesley Publishing Company.

[Lientz and Swanson ()] *Software Maintenance Management*, Lientz , E Swanson . 1980. Addison-Wesley.

[Arnold ()] *Software Reengineering*, Robert S Arnold . 1993. Alamitos, California: IEEE Computer Society Press Los. p. 60.

[Arnold ()] *Software Reengineering*, Robert S Arnold . 1994. Los Alamitos, California: IEEE Computer Society Press.

[Standard Glossary of Software Engineering Terminology ()] *Standard Glossary of Software Engineering Terminology*, IEEE Std. 610.12. 1990. Los Alamitos, CA: IEEE Computer Society Press.

[Valenti ()] *Successful Software Reengineering*, Sal Valenti . 2002. 1331 E., Chocolate Avenue, Hershey: IRM Press.