*International Journal of Engineering Research and Development*
***e*-ISSN: 2278-067X, *p*-ISSN: 2278-800X, www.ijerd.com**
*Volume 3, Issue 7 (September 2012), PP. 01-06*

# Metric Based Approach to Find Maintenance, Reengineering and Retirement Need of Software with a Case Study

Sumesh Sood[1], Arvind Kalia[2], Hardeep Singh[3]
[1,2]*Department of Computer Science, Himachal Pradesh University, Shimla, Himachal Pradesh, India.*
[3]*Department of Computer Science & Engineering, Guru Nanak Dev University, Amritsar, Punjab, India.*

***Abstract—A significant collection of software assets are used in all high-tech organizations and software business. Today, the major concern is with the maintenance and reuse of software which can be further invested in such a manner that its value may increase with passage of time but not like accountability whose value diminishes with the time. It has been an issue of forceful debate and keen interest for more than two decades for the software engineers to make a choice between the software reengineering and software maintenance. Metric structure which has been discussed in this work can be helpful in calculating the required reengineering cost of system and its various modules. Its authenticity has been proved with the help of a case study. A choice can be made regarding the maintenance, reengineering and retirement need of software or parts of software after obtaining results by these metrics.***

***Keywords—Partial Reengineering, Reengineering Requirement Cost, Reengineering Requirement Cost of Module, Software Reengineering, Software Maintenance.***

## I. INTRODUCTION

Reengineering is the examination of the design and implementation of an existing legacy system and applying different techniques and methods to redesign and reshape that system into hopefully better and more suitable form. Hence software reengineering is the examination, analysis and alteration of an existing software system to reconstitute it in a new form, and the subsequent implementation of the new form [1].

But sometimes it is more cost effective not to reengineer the whole software but the part of the software. This process is called partial reengineering. The process of partial reengineering provides an opportunity to look at existing design and to identify opportunities for improvements. Hence, Partial Reengineering is the process of identifying parts of the system to be changed (candidate system), creating an abstract description of a system, reason about a change at the higher abstract level, re-implements the candidate system and integrate the whole system i.e. old system (excluding candidate parts) and redeveloped candidate system [2].

### A. Requirement for Reengineering

Most of the information systems have been in use for a long time. With the passage of time there may be a requirement to change the system i.e. need to maintain the system (corrective, adoptive or perfective maintenance). But as the changes are made in the system there is possibility that structure of the system degrade, hence error rate increases which makes the chances more expensive. With each change some errors may be added in the system, which may affect the performance. Hence with each change in the system number of errors in the system is likely to be increased, which increases the cost to implement further changes as shown in Figure 1. As this cost increases there is a need either to discard the whole system (Retirement) or to rebuild the system with little change in design and implementation techniques. This process is called reengineering [3].

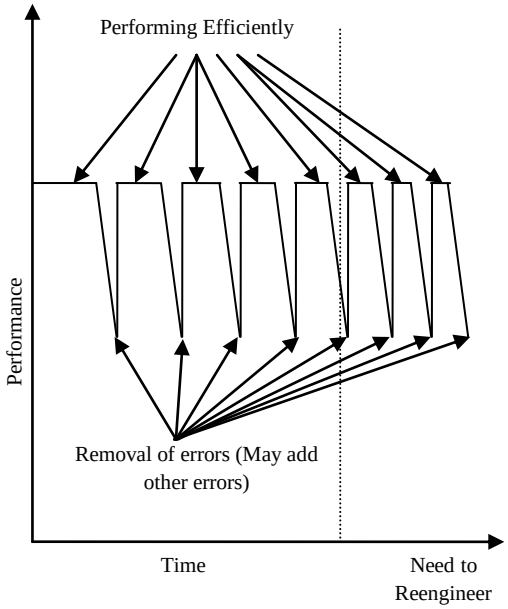


***Figure 1:*** Effect of Time on Performance of System

Due to two main reasons the companies go through the process of reengineering are technology changes and staying one step ahead of the competition [4]. Software reengineering plays a very important role in keeping high software maintenance costs under control, recovering existing software assets, and establishing a base for future software evolution. The idea is to improve or transform, existing software so one can easily understand, control, and use it anew. Software engineering is integral part for achieving many goals in software maintenance and for planning for change in existing system.

**B. Software Metrics and Reengineering**

Software metric is a measure of some property of a piece of software or its specifications. Usually measurements are made to provide a foundation of information upon which decisions about which software engineering task can be planned and performed better [5]. Software metrics support various reengineering tasks also. Software metrics helps in finding the useful and useless components of the software. They help in understanding of the legacy system and can often find hints about design flaws that are obstructing the modification and the extension of the system. With the help of metrics one can estimate the cost of reengineering process and compare it with new software development. Metrics can help to provide easy access to meaningful information about the source code without requiring the software engineer to read through the entire source code. Instead information given by metrics can be used to make a more efficient study of the source code based on the points of interest indicated by the metrics result.

## II. REVIEW OF LITERATURE

In early years of the information revolution the need for reengineering was not acknowledged by the wider community, instead, attention was directed towards the new ways of creating better software. In 1990 Chikofsky and Cross described the reengineering of software as 'the examination and alteration of a subject system to reconstitute it in a new form and the subsequent implementation of the new form' [6].

The fact, at that time so much attention was given to reengineering and software reuse that entire businesses were caught up in the excitement and had their entire business structure recognized according to the newly developed reengineering methodologies and patterns that had emerged.

Soon it became apparent that the reengineering of both business and its software was not as easy as the consultants had first believed. Over half of the reengineering processes of the time failed, mostly due to inexperience and lack of customer's involvement. With these failures resulting in huge lose for the companies and soon the reengineering boom was over and so was the interest in reengineering development. This shows that reengineering, the reorganization and redesign of a system is very important, since if these costs can be reduced, much will be gained for software users.

In 1998 Unified Modeling Language was adopted unanimously by the membership of OMG (Object Management Group) as a standard [7].

MORALE (Mission Oriented Architecture Legacy Evolution) is reengineering methodology developed in Georgia Institute of Technology in 1997 by Abowd *et al.* The MORALE address the problem of designing and evolving complex software systems [8].

In the late 90's Kazman, R. [9] and Woods, S. [10] developed a conceptual "horseshoe" model that distinguishes different levels of reengineering analysis and provides a foundation for transformations at each level, especially for transformations at the architectural level. This model describes the rich set of technical choices that reengineers make. However, because of its technical focus, it has not been accessible to decision makers in a form that can assist them in deciding on complex options regarding the future of their legacy systems.

Singh and Sood (2006) in their study presented four scenarios of software reengineering [4] and with this the need for partial software reengineering arises.

In 2007 Chiang in his work explained the connection between the stability modeling and reengineering process for legacy system [11].

In 2009 Mishra *et. al.* have designed a model CORE (Component Oriented Reverse Engineering) to identify and develop reusable software components [12]. By using the reverse engineering techniques they extracted architectural information and services from legacy system and later on convert these services into components that can be reusable later. But again in this paper they have explained reengineering of the software. But they were not able to explain that why they had preferred to reengineering the components, in place of developing the new components from scratch.

In 2010 Tucker and Simmonds presented a paper in Seventh International Conference on Information Technology [13]. In this paper they describe a case study in perfective and adaptive reengineering. But again there was no comparison why they choose reengineering in place of maintenance or developing software from scratch.

## III. RESEARCH METHODOLOGY

In this work source code of software has been downloaded from the web site http://www.sourcecodesworld.com/source/show.asp?ScriptId=1221 is taken as case study. A metric framework has been used to calculate reengineering requirement cost (RRC) & reengineering requirement cost of module (RRCM). On the basis of the results obtained by these metrics, a decision can be made regarding maintenance / reengineering / retirement need of the software / part of software.

## IV. COST OF REENGINEERING

To calculate the requirement for reengineering four variables Defect Cost (*DC*) [14], Fault Cost (*FC*) [14], Reengineering Requirement Cost (*RRC*) [14] and Reengineering Requirement Cost of Module (*RRCM*) [14] are used.

*DC* can be calculated by multiplying ratio between defect age in year and software age in year with lines of code affected by defect and total no of lines of codes. *DC* is directly proportional to defect age, as the age of defect increases, cost of *DC* increases, because it may be possible that this defect is difficult to remove. Also *DC* is inversely proportional to the age of software, as the software is older that means defect is not severe, as it has not affected the software for long time. Hence its cost decreases. Defect Age and Software Age are taken as upper integer in years because if software is new then this ratio do not fluctuate and remain one. *DC* also depends on ratio of number of lines of code affected by defect to total number of lines of codes. If there is requirement to add other functionality then lines affected by defect is number of statements that will be added for increasing the functionality and can be calculated by using Fuzzy Logic Method [15].

*FC* can be calculated by calculating ratio between mean time to maintenance and time between last two maintenance tasks. As failures become frequent above ratio increases and hence value of *FC* increases.

The value of *RRC* can be calculated by adding twenty times value of *DC* of each bug (number of bugs can be calculated by the metric Number of Defects [16]) and *FC*.

$$DC = \frac{Ceil\,(Defect\ age\ in\ years\ )}{Ceil\,(Software\ age\ in\ year\,)} \times \frac{Lines\ affected\ by\ defect}{Total\ number\ of\ lines}$$

$$FC = \frac{Mean\ time\ to\ maintenance}{Time\ between\ last\ two\ maintenance}$$

$$RRC = 20 \sum_{i=1}^{n} DC_i + FC \qquad \textbf{(i)}$$

To find whether there is requirement to maintain, reengineering or retire the system or its modules, two cases arise.

### A. Case 1.

Software consists of one module only.

a. If *RRC* is less than 3.0 then there is no requirement of reengineering.
b. If *RRC* is between 3.0 and 6.0 then there may be requirement for reengineering.
c. If *RRC* is between 6.0 and 10.0 then there is high requirement for reengineering.
d. If *RRC* is greater than 10.0 then reengineering cost will be very high and system must retire and redesign using new architecture and techniques.

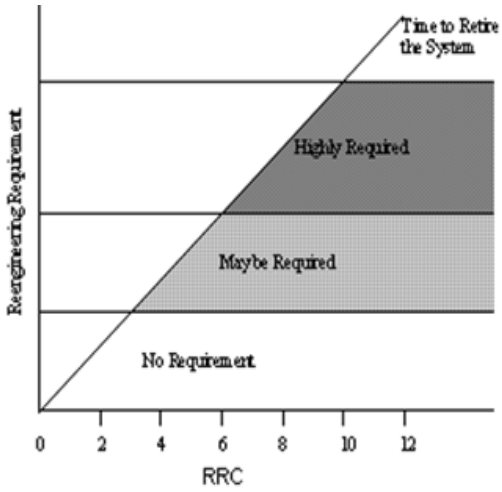


Figure 2 Relationship between RRC & Reengineering Requirement

**B. Case 2.**

If software consists of more than one module and if the value of *RRC* is between 3.0 and 6.0 then each module of the software is checked by adding half of *RRC* of whole software to *RRC* of $i^{th}$ module to calculate Reengineering Requirement Cost of Module i (*RRCM*).

$$RRCM_i = \frac{1}{2} \quad RRC + RRC(M_i) \qquad \text{(ii)}$$

a. If *RRCM* is less then 6.0, then there is no requirement of reengineering.
b. If *RRCM* is between 6.0 and 10.0, then there is requirement to reengineer the module.
c. If *RRCM* is greater than 10.0 then reengineering cost will be very high and new module should be redesign.

## V. CASE STUDY

In this work source code of software (TicTac1.c) have been downloaded from the web site "http://www.sourcecodesworld.com/source/show.asp?ScriptId=1221" is taken as case study having 217 LOC. This is a simple TicTacToe game program build without graphics.

On analyzing the program, it was found that it is difficult to understand and modify because the whole program i.e. starting menu, drawing entry screen (in which players enters their name, choose their moves), verifying correct moves, logic to find win/draw and last screen (showing result) are all included in one program.

After studying this program it was found in the same module all the components of the program has been enclosed. Hence it has low cohesion. Internal logic to find win/draw should be in separate module and entry screen and messages should be in separate module, so that the program should be easy to understand and maintain/change.

## VI. ANALYSIS

**i.** TicTac1 Software discussed in the section V has 217 LOC and out of these 217 lines 81 lines have to be shifted in the other module. When calculated, its DC is 0.3732 and FC is 0.0. Hence RCC is 7.465. Comparing this value with the Figure 2, the whole program should be reengineered. Since value of RRC is above 6, Case 2 of Section III (calculation for RRCM) does not arise.

**ii.** Now new software (TicTac2) has 220 LOC. If this software has to be converted in mouse driven software, then total lines needs to be changed are 155. After applying formulae 1, 2, 3 of Section III respectively, DC value of the software is 0.704 and FC is 0.0. Hence RCC will be 14.08. Comparing this value with Figure 2, the program should be retired and redesigned from scratch.

Also when authors try to convert TicTac1 software into TicTac2 (consist of two modules), some changes has to be made in basic program, but rest of the structure of the software remains the same. But when authors try to convert TicTac2 software into third software (TicTac3), then almost complete basic structure and methodology have been changed, other than both software consist of two modules one for input/output operations and other for logic of the program. To analysis the difference between TicTac1, TicTac2 and TicTac3 a tool SourceMonitor downloaded from the website is used. After applying the SourceMonitor on TicTac1, TicTac2 and TicTac3, results obtained are shown in table I.

From the table I it is clear that though number of statements in TicTac2 is more than TicTac1 and TicTac3 is more than TicTac2, but average complexity decreases in TicTac2 and further in TicTac3. Hence software becomes less complex. Figure 3 and 4 shows graphical representation of table I.

***Table I:*** Result of Tool SourceMonitor on TicTac1, TicTac2 and TicTac3

| Software | Statements | % Branches | Functions | Avg Stmts/Function | Max Complexity | Max Depth | Avg Depth | Avg Complexity |
|---|---|---|---|---|---|---|---|---|
| **TicTac1** | 207 | 21.3 | 6 | 54.2 | 29 | 7 | 2.75 | 12 |
| **TicTac2** | 220 | 20 | 8 | 41.1 | 29 | 4 | 1.8 | 9.25 |
| **TicTac3** | 284 | 16.9 | 12 | 37.5 | 29 | 4 | 1.91 | 7.08 |

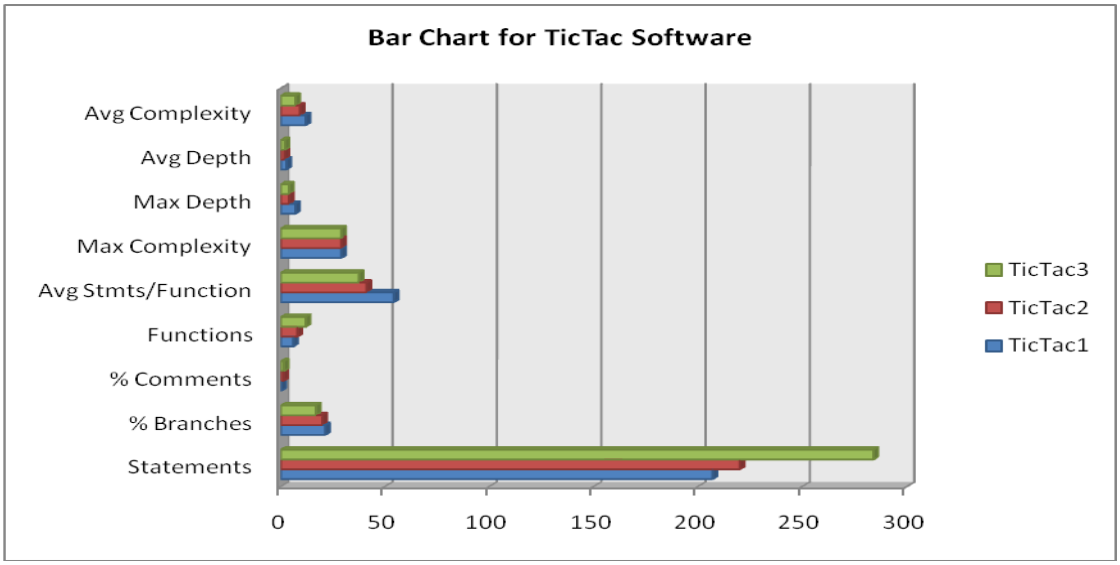


***Figure 3:*** Bar Chart for TicTac Software

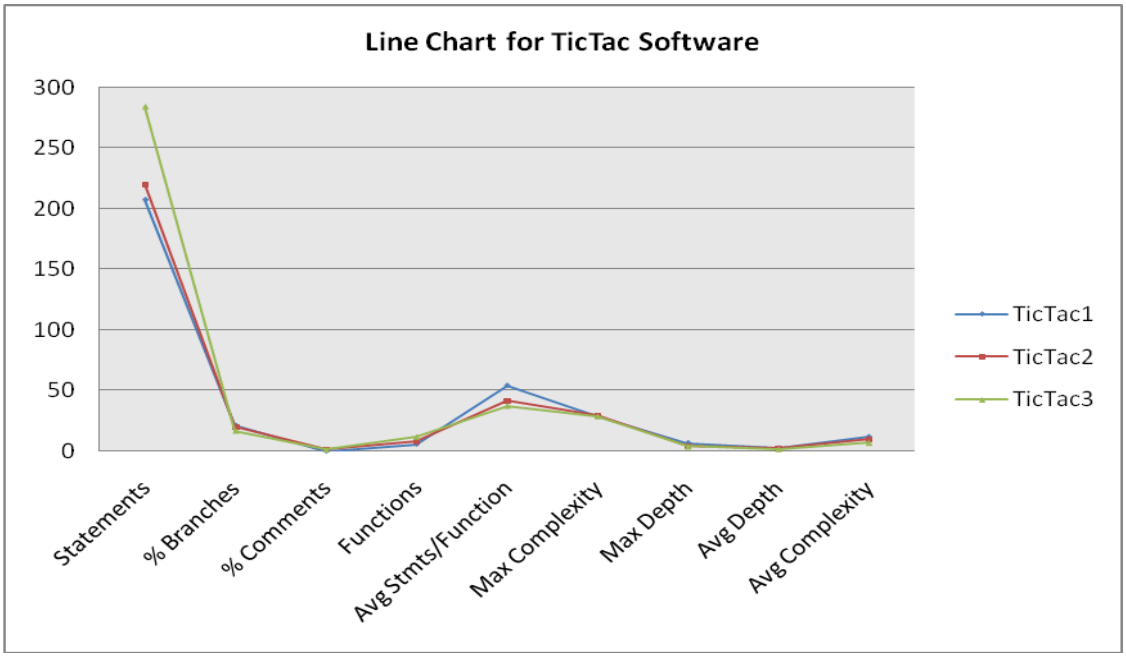


***Figure 4:*** Line Chart for TicTac Software

## VII. CONCLUSIONS & FUTURE DIRECTION

As the requirements of users get changed with passage of time, it becomes mandatory to make changes in software. The new changes may results with the introduction of new bugs in the system. After some time it requires regular maintenance. To find whether software can be maintained or there is requirement to reengineer the software, two metrics are used in this work with case study of Tic Tac Toe game. Software reengineering metrics (RRC and RRCM) can be used to calculate reengineering requirement cost of entire software and reengineering requirement cost of each module of the software respectively. The results obtained after applying the proposed metric become the basis for decision of maintenance,

reengineering or retirement of software/ part of software. This study has indicated that metrics can be useful tools to in the re-engineering operations.

On the other hand, it should not be forgotten that metrics will never be able to offer 100% precise results. Metrics will always be useful hints, but never firm certainties. A human mind will always be necessary to take the final decision in re-engineering matters. It is also strongly necessary to let the conclusions of this study be validated by more experimental studies made on other large scale software.

## REFERENCES


[1]. Manzella, J. and Mutafelija, B., *Concept of Reengineering Life Cycle*, (Proceeding of the Second International Conference on Systems Integration, Morristown, NJ), pp. 566-571, Jan, 1992.

[2]. Singh H., Sood S. and Kalia A., *Use of Metrics in Reengineering Process*, Ultra Scientist of Physical Sciences (International Journal of Physical Sciences), Vol. 19 No. 1, pp.57-64, 2007.

[3]. Singh H., Kalia A., Sood S., Kaur P. and Kaur K., *A Metric Framework for Determination of Requirement for Reengineering*, The ICFAI Journal of System Management, Vol. 6 No. 2, pp.37-43, 2008.

[4]. Singh, H. and Sood, S., *Reengineering Processes and Methods: A Study*, (Proceedings of the Conference Innovative Application of IT and Management for Economic Growth of India), Jalandhar, India, pp. 392-404, Mar, 2006.

[5]. Ducasse, S. and Demeyer, S., *The FAMOOS Object-Oriented Reengineering Handbook*, (University of Bern), 1999.

[6]. Chikofsky E. J., and Cross J. H., *Reverse engineering and Design Recovery: A Taxonaomy*, IEEE, Volume 7, Issue 1, p.15, 1990.

[7]. Framinghan, M. A., *Unified Modeling Language Specification Version 1.1*, Object Management Group. Internet www.omg.org, 1998.

[8]. Abowd G., Goal A., Jerding D. F., McCracken M., Moore M., Murdock J.W., Potts C., Rugaber S. and Wills L., *MORALE, Mission Oriented Architectural Legacy Evolution*, (Proceeding of International Conference on Software Maintenance, Bari, Italy), 10/01/97-10/03/97pp 150-159, 1997.

[9]. Kazman R., Woods S. G., and Carriere S. J., *Requirements for Integrating Software Architecture and Reengineering Models: CORUM II*, (Proceedings of WCRE 98, Honolulu, HI), pp 154-163, 1998.

[10]. Woods S., Carriere S. J. and Kazman R., *A Semantic Foundation for Architectural Reengineering and Interchange*, (Proceedings of the International Conference on Software Maintenance (ICSM-99) Oxford, England), pp. 391-398, 1999.

[11]. Chiang C. C., *Software Stability in Software Reengineering*, (IEEE International Conference on Information Reuse and Integration, Las Vegas), 13/08/07-15/08/07, pp. 719-723, 2007.

[12]. Mishra S.K., Kushwaha D.S. and Misra A.K., *Creating Reusable Software Component from Object-Oriented Legacy System through Reverse Engineering*, Journal of Object technology, Jan-Feb 2009, pp 133-152, 2009.

[13]. Tucker, D. C. and Simmonds, D. M., *A Case Study in Software Reengineering*, (Proceeding of Seventh International Conference on Information Technology, Las Vegas, Nevada, USA), April 12- April 14, ISBN 978-0-7695-3984-3, pp 1107-1112, 2010.

[14]. Singh H., Sood S., Kaur R. and Ratti N., *Metric Framework for Reengineering Process*, Punjab University Research Journal. Vol. 57, pp.251-255, 2007.

[15]. Putnam, L. H. and Myers, W., *Measure for Excellence: Reliable Software on Time, within Budget*, (Englewood Cliffs, NJ), 1992.

[16]. Konda, K. R., *Measuring Defect Removal Accurately*, (The Enterprise Development Conference), p 35, 2005.