Metric Framework for Software Reengineering Process to Improve the Efficiency with a Case Study

Dr. Sumesh Sood
Professor
Department of Computer Engineering
IKGPTU Dinanagar Campus, Dinanagar, India

Abstract— The reengineering process is used to transform or improve existing software so that it can be understood, controlled and reused as new software. Software metric can help the reengineering process to make the process easy, economical and efficient. 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. In this paper, a metric framework has been proposed for reengineering process to make the reengineering process easy, economical and efficient. In the framework, metrics have been selected that can be used to find the cost of upgradation of software. On the basis of result obtained, a decision can be made regarding maintenance / reengineering / retirement need of the software / part of software. Metric set is also used to find which parts of the software require reengineering. A case study has been undertaken to validate this metric framework.

Key words: Software Reengineering, Partial Reengineering, Maintainability Index, Source Monitor

I. INTRODUCTION

Most of the software system starts off in a well-designed state, in the meantime requirements evolve and customer demands new functionality, very often in such a way that the original design did not anticipate. At that time, a complete redesign may not be practical, and a system is bound to gradually lose its original well-defined structure. Then there is a need to restructure the software. This re-structuring is often referred to as either refactoring or reengineering. While main goal of refactoring is to maintain the behaviour of the code being changed, it is very often the case that new functionality is added (or old functionality changed) during a reengineering process. Reengineering very often necessitates some sort of refactoring, but refactoring very rare impact on the reengineering process. Software metric is a measure of some property of a piece of software or its specifications. Software metric are used in reengineering process to make the process easy, economical and efficient. 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. In this paper metric set based approach for software reengineering is presented. A case study of software has been undertaken to validate this metric based approach.

II. NEED FOR SOFTWARE REENGINEERING

Most of the software system have been developed using the system development methods, programming tools, data base handler etc. that are available when the work of development of system starts. With the passage of time environment changes and with change in environment these systems require some changes i.e. need to maintain the system (corrective, adoptive or perfective maintenance). But as changes are made in the system there is a chance that structure of system degrades, hence error rate increases which make the further changes more expensive. With each change, we may add some error in the system which may affect after some time. Hence with each change in the system number of errors in the system increases, this in turn increases the cost to implement further changes. As this cost increases, a need arises either to discard the whole system (retirement) or to transform the existing system into a new form with little change in design and implementation technique. Later is called reengineering. The reengineering is required when

- The system changes affect a subsystem and the subsystem needs to be redesigned.
- Hardware and software support has become outdated and obsolete.

III. REVIEW OF RESEARCH AND DEVELOPMENT IN THE SUBJECT

A. International Status

In the earlier days of information revolution the need for reengineering was not acknowledged by the wider community. Instead, attention was directed towards the discovery of new ways of creating better software. Almost no attention was given to the old systems that were getting more and more outdated. In addition, the businesses were changing rapidly and with them came the need for appropriate information software. It always seemed like newly developed software were not good enough to accommodate the business needs. At that time this was called software shortage.

Lehman and Belady (1985) in their study, organized notation of software evolution at IBM [1] and has since progresses to become distinct field of research. Then, in the early 90's the focus of the system development changes very rapidly from the development of new software to reengineering of legacy systems.

In 1990 Chikofsky and Cross described the reengineering of software as 'the examination and alteration of a subjected system to reconstitute it in a new form and the subsequent implementation of the new form' [2]. The fact, that so much attention was given to reengineering was due to that the entire businesses were caught up in the excitement and had their entire business structure recognized according to the newly developed reengineering methodologies and patterns were emerged.

Soon it became apparent that the reengineering of both business and its software was not as easy as the consultants 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, the reengineering boom was over and so was the interest in reengineering development. This shows that reengineering, reorganization and redesign of

a system is very important, only if these costs can be reduced, much will be gained for software users.

In the late 90's Kazman [3] and Woods [4] developed 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.

In 2007 Chiang in his work explained the connection between the stability modeling and reengineering process for legacy system [5].

In 2009 Vintila and Palaghita gives a formula to estimate the cost of reengineering process [6]. The formula is:

$$\text{Cost} = \sum_{i=1}^{n} S_{m_i} * C_{m_i} * \text{ra}$$

Where

n is number of modules the application has,

S_{m_i} is size of module i; this gets values reporting the size of the module with the medium set size,

C_{m_i} is complexity of module i; this gets values reporting the module of the module with the medium set complexity,

ra has the value 1 if the module is affected by the reengineering process and has the value 0 is the module is not affected.

In the formula they proposed that the cost of reengineering depends upon cost of affecting modules. But reengineering requirements also depends upon many more factors like frequency of maintenance, coupling with other modules, cohesion within the module, numbers of defects in the program, lines of program affected by the defects etc. So to calculate requirement of reengineering a lot of more factors should also be needed to be incorporated.

In 2010 Tucker and Simmonds presented a paper in Seventh International Conference on Information Technology [7]. 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.

In 2011 Hong Zhou proposed knowledge based software reengineering approach in the context of software reengineering and knowledge representation [8]. It is an application of description logic and ontology to the task of constructing computable models for the software reengineering domain. His thesis aims to improve the traditional software reengineering methods by proposing a knowledge based software reengineering approach via ontology and description logic.

Cholakov and Birov (2012) in their article represented a model for automated reengineering of legacy software systems [9]. It describes in details the processes of software translation and refactoring and the degree of automation that these processes may achieve.

In 2016 Graciamary and Chidambaram proposed enhanced reengineering mechanism to minimize the complexity in software reengineering process [10]. This mechanism was proposed to reduce the cost and time.

B. National Status

Singh and Sood (2006) in their study 'Reengineering Process and Methods: A Study' analyzed various reengineering methods and presented four scenarios of software reengineering [11].

Jasmine and Vasantha (2009) explains the goals of reengineering test management and how to achieve it and the approach as demonstrated, constructs useful models that act as predictors of testing effectiveness in component based enterprise applications [12].

In 2009 Mishra et. al. has designed a model CORE (Component Oriented Reverse Engineering) to identify and develop reusable software components [13]. 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. Again in this paper they explained reengineering of the software, but they were not able to explain that why they preferred reengineering the components, in place of developing the new components from scratch.

Asit Kumar Gahalaut and Padmavati Khandnor (2010) in their paper 'Reverse Engineering: An Essence for Software Reengineering and Program Analysis', explained that to affect change management, even a simpler upgrade may become diffult [14]. This is the reason why eminent development houses now focusing on advanced documentation support. Re-engineering code environment thence largely affect the problem issues regarding program comprehension when the software size grows enormously. Reverse Engineering is a methodology that greatly reduces the time, effort and complexity involved in solving these issues providing efficient program understanding as an integral constituent of re-engineering paradigm.

Sood, Kalia and Singh in 2012 explained metric based approach for software reengineering [15]. Result obtained after applying this approach, decision can be made regarding the maintenance, reengineering and retirement need of software or parts of software.

In 2012 Tarar and Kumar describes traditional reengineering and discusses the emerging process of hybrid-reengineering which is often used as means to simplify the cumbersome tasks [16]. Hybrid re-engineering is a reengineering process that uses not just a single, but a combination of abstraction levels and alteration methods to transition an existing system to a target system. The paper represents how maintenance is going to be effect with the help of given software engineering approaches. An analysis of various possible risks, their impact and mapping with various attributes is correspondingly depicted. Their paper is also presenting the way to reduce the impact of most of these risks by using hybrid re-engineering.

Methakullawat and Limpiyakorn (2014) presented an approach to extracting the architectural design from the archaic PL/SQL code as the initiative of legacy transformation for software modernization [17]. The pro-totype system is implemented to facilitate the transformation of PL/SQL code modeled with four main sections (Form, Menu, Report, and Library) to the metamodel of UML Class dia-grams that would support the comprehension of software architecture.

Panwar and Saini in 2015 proposed a new model for quality of Hybrid-re-engineered product by using some

matrices for all the different tracks of Hybrid Re-engineering to make effective Hybrid Re-engineering process [18].

From the review of the related studies, it can be concluded that for the last 25 years software reengineering has become an important field of computer science and an active field of research. However, a few studies discussed the circumstances where reengineering is preferred over maintenance, how metrics can help in software reengineering process and the various parameters on the basis of which the partial reengineering may be helpful to maintain the legacy software. So, stress is to be given to study and discuss the need of partial reengineering and use of software metrics to reduce the effort and cost, and to increase the efficiency during reengineering of a legacy system.

IV. RESEARCH METHODOLOGY

A metric framework has been proposed for reengineering process. This framework is used in different phases of proposed model to make reengineering process easy, economical and efficient. Metric framework has been used to find the cost of reengineering process. In this framework, metrics have been selected that can be used to find the cost of upgradation of software. On the basis of result obtained, a decision can be made regarding maintenance / reengineering / retirement need of the software / part of software. Metric set is also used to find which parts of the software require reengineering. The parts which require reengineering is called candidate system. A case study has been undertaken to validate this metric framework.

To make reengineering process automated and easy, following tools are used:

A. SourceMonitor

To analysis the difference between old software and new software a tool SourceMonitor downloaded from the website http://download.cnet.com/SourceMonitor/3000-2383_4-10222336.html is used. SourceMonitor is a tool that measures and records source code metrics with persistence and historic comparison.

B. Line Counter

Line Counter is a command-line tool for software developers. Line Counter is always available for download from this address: http://noeld.com/programs.asp?cat=misc#lcounter. It reports the total number of lines of code and comment lines in C, C++, C#, Java, JavaScript and PHP source files

V. METRICS FOR REENGINEERING PROCESS

In this work Rainfall model [19] is used for reengineering process. Following metrics that are found to be useful in partial reengineering process are used:

- Defect Age (DA) [20]
- Lines of Code (LOC) [21]
- Tight Class Cohesion (TCC) [22]
- Coupling between Objects (CBO) [22]
- Maintainability Index (MI) [23]
- Defect Cost (DC) [24]
- Fault Cost (FC) [24]
- Reengineering Requirement Cost (RRC) [24]
- Reengineering Requirement Cost of Module (RRCM) [24]

VI. CASE STUDY

Software (name SalaryOld) used in an organization in Pathankot to manage salary of all the employees in the organization is taken as a case study. Software consists of 30 forms, 2 methods and 20 reports. Software is used to calculate salary of the employees of various units in different months with different DA, HRA, CA, EPF, advance, LIC deduction, arrears etc. But it has no criteria to calculate bonus, which is calculated manually. So there is requirement to add criteria to calculate bonus of all the employees and display reports to the employee related with bonus. One more drawback of the software is that there are only two modules, one contains methods and other contains variables. All the methods used in the software are packed in single module 'method' without any logic, which make it less cohesive. Also there is no documentation, so some documentation is required, which may help during maintenance of the software. Hence, there is need to change the system by adding forms and reports for bonus, breaking the module 'method' logically and adding documentation.

A. Phase I: Identification of Candidate System

In software of 30 forms and 2 modules, there is a need to add 5 forms and 5 more forms requires modifications to add criteria to calculate bonus. There is also requirement to break the module 'methods' into 5 modules, so that modules implements separate concepts. When calculated its *DC* value (metric 6), it come out to be 0.4688 (15/32). Since software is changed first time, so *FC* value (metric 7) of the software is 0.0. Now calculating the value of *RRC*, the value of *RRC* (metric 8) is 9.376. So, there is requirement for reengineering. Since value of *RRC* is above 6, hence this software does not need partial reengineering and complete software should be reengineered as a whole and requirement to calculate *RRCM* (metric 9).

B. Phase II: Reverse Engineering

Now in the software some modules are more complex and are tightly coupled with other modules. Reverse engineering can be started from these components. In SalaryOld two forms FRMEMPMASTER.frm and FRMSALMODI.frm and a module METHODS are candidate for important components. *LOC* values (metric 2) of these components are 1693, 1418 and 259 respectively and coupling values (metric 4) are 8, 7, 12 respectively. To understand how the system works, emphasis should be on understanding these three components and their interactions by studying their source code. Hence, making them a starting point during reverse engineering process.

C. Phase III: Architecture Transformation or Change

After reverse engineering a module METHODS with highest coupling value 12 (metric 4) is broken into five parts to reduce the coupling and increase the cohesion. It is not possible to break the modules FRMEMPMASTER.frm and FRMSALMODI.frm into parts though it has high value of coupling, but they implements logical cohesion. Also parts related to bonus are added in the software. By adding criteria of bonus, statements of the software increases (11882 to 12863), but average complexity decreases (20.625 to 17.487).

D. Phase IV: Development of Candidate System

In the redevelopment of candidate system Fuzzy Logic Method has been used to estimate the size of the software. This method uses LOC (metric 2) to approximate the size of new software. Since, in place of 32, now the software has 41 components, so new software has approximately 11882*41 /32 = 15223 LOC.

Now changes are made in the design and are implemented in the coding. To make sure system does not misbehave after the coding changes, the sensitive parts have been identified (metric 2 and metric 4). In SalaryOld two forms FRMEMPMASTER.frm and FRMSALMODI.frm and a module METHODS have highest coupling values, hence these three components are thoroughly tested after coding. Also in the new software cohesion has been reduced by breaking the component METHOD into five parts.

Three modules FRMSALPROCESS, MDIMAIN and METHODS have highest coupling value and are mainly affected by the addition of bonus part. Hence these three modules are thoroughly tested after coding.

Software	Functions	Vocabulary	Cyclomatic Complexity	Average LOC	Comments	Maintainability Index
SalaryOld	32	141.5842448	20.625	108.40625	0.562234272	65.15208169
SalaryNew	41	137.1686606	17.48780488	97.31707317	1.865402443	69.08972275

Table 2: Maintainability Index of SalaryOld and SalaryNew

E. Phase V: Integration

Since the whole software is reengineered in this case, so there is no need for integration and the software is tested for complexity (using SourceMonitor) and maintainability (metric 5). It is found that average complexity of the software is decreased from 20.625 to 17.487 and value of Maintainability Index is increased from 65.15 to 69.09.

In this paper, to differentiate Salary software before and after reengineering, they are named as SalaryOld (before reengineering) and SalaryNew (after reengineering) respectively.

To analysis the difference between SalaryOld and SalaryNew a tool SourceMonitor is used. Applying the SourceMonitor, it has been found that although LOC of SalaryNew (12863) is more as compare to SalaryOld (11882), but average depth decreases from 3.28 to 3.13 and average complexity decreases from 20.6 to 17.5, as shown in Table I.

Fig. 1 and Fig. 2 are showing pictorial representation of Table 2.

Software	Files	Statements	% Branches	Subroutines	Biggest Subroutine	Max Depth	Avg. Depth	Ave Complexity
SalaryOld	32	11,882	3.7	568	125	9+	3.28	20.625
SalaryNew	41	12,863	3.7	595	160	9+	3.13	17.487

Table 2: Result of Tool SourceMonitor on SalaryOld and SalaryNew

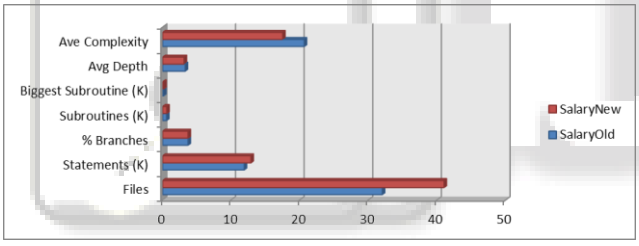

Fig. 1: Bar Chart for SalaryOld and SalaryNew Software

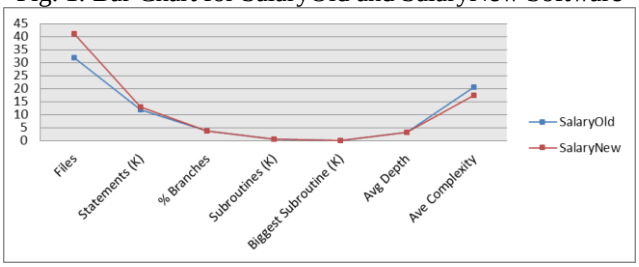

Fig. 2: Line Chart for SalaryOld and SalaryNew Software

Now to find effect of reengineering on the maintainability of the software, maintainability index of the software is calculated. Table II shows the value of maintainability index of SalaryOld and SalaryNew.

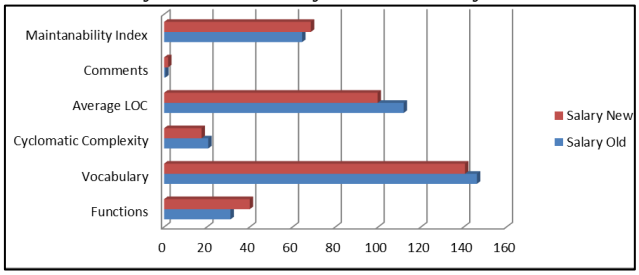

Fig. 3: Bar Chart for Maintainability Index of SalaryOld and SalaryNew Software

Fig. 3 and Fig. 4 showing pictorial representation of Table 2.

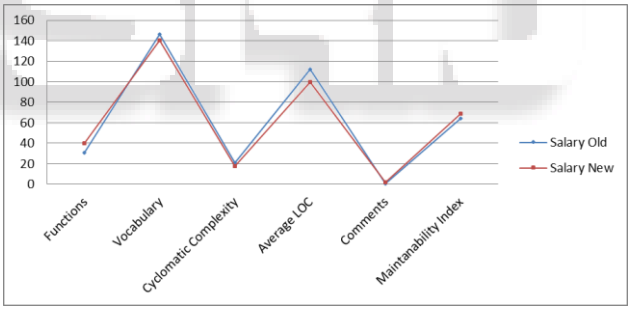

Fig. 4: Line Chart for Maintainability Index of SalaryOld and SalaryNew Software

From the Table II it can be seen that though number of functions are increasing from 32 to 41 in SalaryNew as compare to SalaryOld, but vocabulary is decreasing from 141.58 to 137.16, cyclomatic complexity is decreasing from 20.6 to 17.5 and average LOC is decreasing from 108.4 to 97.3 and hence, maintainability index increases from 65.15 to 69.09. So, SalaryNew is more maintainable as compare to SalaryOld.

VII. CONCLUSION & FUTURE DIRECTION

This study has indicated that metrics can be used to in the re-engineering operations. The metric based approach discussed in this paper, can be used in reengineering to make the process easy, economical and efficient. However, one should remember that metrics will never be able to offer 100% precise results. Metrics always provides 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 necessary to let the conclusions of this study be validated by more experimental studies made on other large scale software.

REFERENCES

[1] Lehman M. and Belady B., Program Evolution, Academic Press, London, p. 9, 1985.
[2] Chikofsky E. J., and Cross J. H., Reverse engineering and Design Recovery: A Taxonaomy, IEEE, 7(1), p. 15, 1990.
[3] 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.
[4] 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.
[5] Chiang C. C., Software Stability in Software Reengineering, Proceedings of IEEE International Conference on Information Reuse and Integration, Las Vegas, pp. 719-723, 13/08/07-15/08/07.
[6] Vintila B. and PalaGhita D., Reengineering of Distributed Collaborative Applications, Journal of Applied Collaborative Systems, 1(2), pp. 90-100, 2009.
[7] 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, pp. 1107-1112, April 12-April 14, 2010.
[8] Zhou H., A Knowledge Based Reengineering Approach via Ontology and Description Logic, Ph. D. Thesis, Software Technology Research Laboratory, De Montfort University, 2011.
[9] Cholakov T. and Birov D., Automated Software Reengineering Model and Framework, Proceedings of the Forty First Spring Conference of the Union of Bulgarian Mathematicians, Borovetz, pp. 225-231, 2012.
[10] Graciamary A. C. and Chidambaram, Enhanced Re-Engineering Mechnanism to Improve the Efficiency of Software Re-Engineering, International Journal of Advanced Computer Science and Applications, 7(11), pp. 285-290, 2016.
[11] Singh H. and Sood S., Reengineering Process and Methods: A Study, Proceedings of the conference Innovative Application of IT and Management for Economic Growth, Jalandhar, India, pp. 392-404, 2006.
[12] Jasmiine K. S. and Vasantha R., Derivation off UML Based Perfformance Models for Design Assesment in a Reuse Based Software Development Approach, Annals. Computer Science Series, 7, pp. 163-180, 2009.
[13] 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, pp. 133-152, Jan-Feb 2009.
[14] Gahalaut A. K. and Khandnor P., Reverse Engineering: An Essence for Software Reengineering and Program Analysis, International Journal of Engineering Science and Technology, 2(6), pp. 2296-2303, 2010.
[15] Sood S., Kalia A. and Singh H., Metric Based Approach to Find Maintenance, Reengineering and Retirement Need of Software with a Case Study, International Journal of Engineering Research and Development, 3(7), pp.1-6, 2012.
[16] Tarar S. and Kumar E., Design Paradigm and Risk Assessment of Hybrid Re-engineering with an approach for development of Re-engineering Metrics, International Journal of Software Engineering & Applications, 3(1), pp. 27-36, 2012.
[17] Methakullawat N. and Limpiyakorn Y., Reengineering Legacy Code with Model Transformation, International Journal of Software Engineering and Its Applications, 8(3), pp 97-110, 2014.
[18] Panwar D. and Saini G.. L., Software Quality Assurance Model for Re-Engineering, International Journal of Advanced Research in Computer Science and Software Engineering, 5(4), pp. 1330-1333, 2015.
[19] Sood S., Use of Metrics for Identifying the Framework of Reengineering Process, M. Phil. Dissertation, MKU, Madurai, 2006.
[20] Morrison J., Software Testing Fundamentals, http://www.softwaretestingfundamentals.com /2009/05/defect-age.html, 2009.
[21] Humphrey W. S., Introduction to the Personal Software Process, SEI Series in Software Engineering, Addison Wesley, p. 22, 1997.
[22] Chidamber S. R. and Kemerer C. F., A Metrics Suite for Object Oriented Design, IEEE Transactions on Software Engineering, 20(6), 1994.
[23] Oman P. and Hagemeister J., Constructing and Testing of Polynomials Predicting Software Maintainability, Journal of Systems and Software, 24(3), pp. 251-266, 1994.
[24] Singh H., Sood S., Kaur R. and Ratti N., Metrics Framework for Reengineering Process, Punjab University Research Journal, 57, pp. 251-255, 2007.

