Analysis of Risks in Re-Engineering Software Systems

Nasir Rashid
Department of CS and I.T
University of Malakand
Pakistan

Muhammad Salam
Department of CS and I.T
University of Malakand
Pakistan

Raees Khan ShahSani
Department of CS and I.T
University of Malakand
Pakistan

Fakhre Alam
Department of CS and I.T
University of Malakand
Pakistan

ABSTRACT
Software re-engineering has become a vital field of computer science and an active research area. The nature of software re-engineering is to improve or transform existing software so it can be understood, controlled and reused as new software. Re-engineering is frequently challenged, because certain risks will threaten the project success. In this article we have described some risks and their classification what we believe to be the most important. From the analysis of risks, some mitigation techniques have been suggested from the existing literature that helps to make the re-engineering projects more beneficial.

Keywords
Software re-engineering, risks, mitigation, analysis.

1. INTRODUCTION
Software Re-engineering is the examination, reorganization, analysis and alteration of an existing software system. It helps to make them more maintainable and to reconstitute it in a new form and the subsequent implementation of the modified system. This process involves the restructuring or recoding of a component or all parts of legacy system without affecting its functionality [1]. Re-engineering is a combination of other processes such as reverse engineering, re-documentation, translation, and forward engineering. The main purpose is to understand the specification, design, implementation of the legacy system and then to implement its modernize form to improve its overall functionality and performance.

The difficulty lies in the conceptual understanding of the legacy system. Usually requirements, design and documentation of programming code is no longer available, or is out of date, so it is not clear to the software engineer that what types of functions are to be shifted. Often the software system contains major functions that are not needed anymore, and those should not be re-coded to the new system [2].

The re-engineering process is not risk free and faces various types of risks as software engineering other approaches face. The risk identification is crucial in development and evolution of a legacy system. Risk identification is very important for effective risk assessment, risk analysis, and management and mitigation of risks. In our proposed work, the potential challenges and risks during transformation are analyzed and then categorized on the basis of severity and nature. A well monitoring technique has been described for the categorized risks. It will help a re-engineering system towards successful and easy maintenance and cost benefit with reduced risk.

2. AN OVERVIEW OF RE-ENGINEERING
Software re-engineering is a technology that involves the examination and alteration of a legacy software system to re-constitute it in a new and modernize form with its subsequent implementation [3]. Software re-engineering is concerned with re-structuring legacy systems to make them more maintainable. Re-engineering may involve organizing and restructuring, re-documenting and recoding the system through suitable and modern programming languages. The functionality and architecture of the system remain the same [4]. Software re-engineering is important to recover and reuse the existing software components, cut off high software maintenance costs under control, and establish a sound base for future software evolution [5].

The main objective of re-engineering as shown in Figure.1, that helps to understand the basic process of transforming a legacy system to a more modernize and up to date software system. The process of restructuring, recoding, redesign and re-documentation is applied on the old and out dated system in order to get the target system according to the new requirements and then re-implement it with old or new functionality with latest available technology.

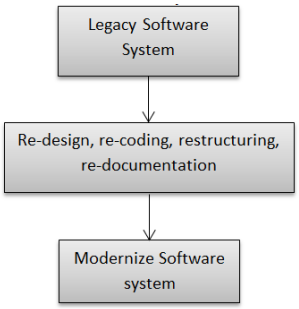

Figure 1: Basic process of re-engineering

Re-engineering a legacy software system has some major advantages over more fundamental approaches to system evolution.

Software re-engineering helps in reducing the cost of writing a system from the scratch by using functionality of the legacy software. According to [2] risks should be considered as an important factor when redeveloping end re-engineering the existing system. Benefits of re-engineering the system can be measured by the formula mentioned below. It is quite possible to derive negative benefit if the cost or risk of re-engineering is sufficiently high compared to the costs of just starting over from scratch.

Re-engineering Benefit =

$$[Old\_Value - (Reeng_{Cost} * Reeng_{Risk}) -$$

$$[New\_Value - (Dev\_Cost * Dev\_Risk)] \quad \text{Equation 1}$$

- It helps to improve the business value of modernize application software by reiterating the critical data and business logic.
- It also helps in increasing system reliability and customer satisfaction.
- Reduces the operational costs.
- Efficient use of the existing staff.
- It improves system performance, accessibility and system documentation.
- It enhances software functionality [6].

3. GENERAL MODEL OF SOFTWARE RE-ENGINEERING

Re-engineering starts with the source code of an existing legacy system and concludes with the source code of a target system. This process may be as simple as using a code translation tool to translate the code from one language to another (FORTRAN to C) or from one operating system to another (UNIX to DOS). On the other hand, the re-engineering task may be very complex, using the existing source code to recreate the design, identify the requirements in the existing system then compare them to current requirements, removing those no longer applicable, restructure and redesign the system (using object-oriented design), and finally code the new target system. Figure 2 depicts a general model for software re-engineering that indicates the processes for all levels of re-engineering based on the levels of abstraction used in software development.

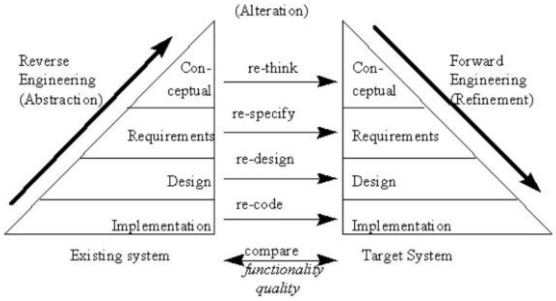

Figure 2: General model for software re-engineering [7]

4. TAXONOMY OF SOFTWARE RE-ENGINEERING

In this section, the following key terms provide a clear scope and taxonomy of the domain of software re-engineering.

Forward engineering (FE) is the traditional process of moving from high-level abstractions and logical, implementation-independent designs to the physical implementation of a system.

Reverse engineering (RE) is the process of analyzing a subject system to (1) identify the system's components and their interrelationships and (2) create representations of the system in another form or higher level of abstraction.

Re-documentation is the creation or revision of a semantically equivalent representation within the same relative abstraction level.

Re-documentation is the simplest and oldest form of reverse engineering and can be considered to be an un-intrusive, weak form of restructuring [8].

Design recovery or *reverse design* recreates design abstractions from a combination of code, existing design documentation (if available), personal experience, and general knowledge about problem and application domains.

Program understanding or *program comprehension* is a term related to reverse engineering. Program understanding implies always that understanding begins with the source code while reverse engineering can start at a binary and executable form of the system or at high-level descriptions of the design [1].

Restructuring is the transformation from one representation form to another at the same relative abstraction level, while preserving the subject system's external behavior (i.e., functionality and semantics) [2].

Recode involves changing the implementation characteristic of the source code. Language translation and control flow restructuring are source-code-level changes [11]

Redesign involves changing the design characteristics. Possible changes include restructuring design architecture, altering a system's data model as incorporated in data structures or in a database, and improving an algorithm [3].

Re-specify involves changing the requirement characteristics. This type of change can refer to changing only the form of existing requirements (i.e., taking informal requirements expressed in English and generating a formal specification expressed in a formal language, such as Z) [4].

5. RISKS IN RE-ENGINEERING SOFTWARE

Software re-engineering is aimed as a mean to mitigate risks and reduce operational and maintenance costs of the legacy software, but legacy transformation have many risks involved as system development projects. The impact of these risks are more severe than the conventional projects development because re-engineering the existing systems involve changes in functional operations that is integral with the current trend and latest available technology to business operations. Early risk identification assists programmers and project managers in preparing for estimation and evaluation of software re-engineering risks and provides a feasible and realistic framework for expectations. Risk identification is essential for effective risk assessment, risk analysis and risk management. These potential risks can be categorized in the following areas as depicted in figure 3 and are discussed throughout this report.

User Satisfaction: Customer satisfaction is an integral key element for any business strategy, therefore it is essential for any business to effectively manage reliable measures for customer satisfaction. In software re-engineering the user satisfaction risks are as below [14].

- Lack of user friendliness
- Budget overflow in un-managed processes
- Unexpected result of the target system
- Unsupported to referential model

Cost: Legacy software is reengineered in order to face the market with latest technology and tools to make it more cost benefitted. Risks involved in cost benefit are mentioned as follow below:

- Less benefit from the cost of re-engineering
- High maintenance cost after re-engineering
- Expensive backup
- High cost to finance report
- Poor quality processes for re-engineering and inconsistency of business plans
- Loss of investments on legacy transformation [15].

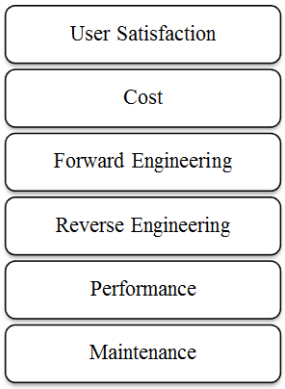

Figure 3: Software Re-engineering Risks

Forward Engineering: Forward Engineering is a traditional technique of moving from high-level abstractions and logical, implementation-independent designs to the physical implementation of a system. Risks involved in FE are listed as follow.

- Captured objects do not integrate to new system
- Difficulty in migrating existing data to for new system
- Degree of preparation for transformation and reverse engineering are not sufficient [16].

Reverse Engineering: *Reverse engineering* (RE) is the process of analyzing a subject software system to (1) identify its different modules and their interrelationships and (2) and to represent the system in another visual form or higher level of abstraction. The RE risk factors are:

- Abstract information cannot be expressed in the designed language for requirements and design specifications.
- It is quite difficult to capture efficient design and few requirements from the source code.
- Existing business knowledge embedded in source code is lost due to inappropriate processes.
- Recovered information is not useful or not used at all.

Performance: The re-engineering process depends heavily on the performance of modernize system. It is the degree of uncertainty that may keep the system to meet its technical specifications or that can result the system not meeting the basic goals. Performance of the new system should definitely be better than the legacy system. The Performance risks factors are listed below.

- Non portability in new system
- Result not matched with the previous system
- Reliability mismatch
- Inappropriate Re-engineering approach and data re-structuring.

Maintenance: Maintenance is an integral factor to be considered in software re-engineering. Decisions are aided by understanding what happens to any software systems over time according to new requirements. The key software maintenance issues can be either managerial or technical or both. The maintenance risks are listed as follow.

- Scheduled Backup
- Recovery of legacy systems
- Improper Re-documentation and data restructuring [17].

We have reviewed a total of 17 papers that were downloaded from different digital libraries, namely Google scholar, cite seer, IEEE Xplore, Springer Link etc. in which risks in software re-engineering have been mentioned from different perspectives. In some papers the risks were categorized in classes according to its frequency and severity in literature and in real world practices. The risks discussed above have been shown in figure 4 according to its frequency in different papers. Some risks have been classified as critical risks due to its negative impact on the consequences of re-engineering projects.

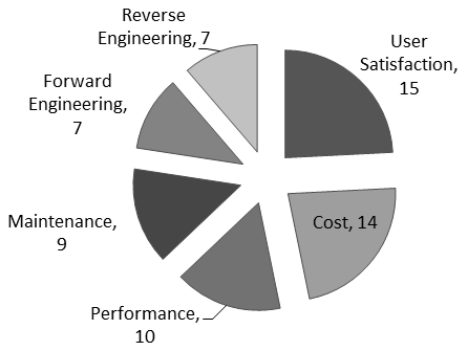

Figure 4: Frequency Distribution of Re-engineering Risks

6. CONCLUSION

Many new software design methodologies and tools have been developed to improve reusability, maintainability and to decrease the cost of development and maintenance. Most companies have software systems that are out of date and costly to maintain. So re-engineering is the best solution to replace the existing software systems. Whenever re-engineering takes places a variety of risks might occur. This review paper has discussed various risks which are classified into different areas and have been shown graphically according to its frequency in different papers. It will help for the proper designing of mitigation plans to mitigate them with respect of its severity and its impact on the re-engineering projects. The main outcome is that organization must adopt new tools and methodologies to make the project with re-engineering according to the customer satisfaction and at a low budget, with current trend and technology. It will lead the project more with good performance and with an ease of maintenance. Further, risks involved in process transformation can be identified, analyzed and contingencies plans can be evolved. An evolutionary method for the mitigation of these risks will be done in a comprehensive manner in my future

work related with process transformation and risk analysis in re-engineering can become effective and efficient.

7. REFERENCES

[1] M.Solvin, and S. Malik. "Re-engineering to reduce system maintenance: A case study",Software Engineering, pp.14-24,2011.

[2] Harry.M.Sneed, "Economics of Software re-engineering", Journal of Software Maintenance, Vol.3, 1991, p.163

[3] Moghaddas, Y., & Rashidi, H. (2009). A novel approach for replacing legacy systems, Journal of Applied Sciences, 9(22), 4086–4090.

[4] J. Ransom, I.S., I. Warren. A Method for Assessing Legacy Systems for Evolution. in Proceedings of the 2nd Euromicro Conference on Software Maintenance and Re-engineering (CSMR'98). 1998.

[5] Detection strategies: Metrics-based rules for detecting design flaws. in Proceedings of ICSM 04 (International Conference on Software Maintenance). 2004. . IEEE Computer Society Press.

[6] Gerardo canfora, Aniello cimitile, "Software maintenance", Journal of Software Maintenance 13(1): 1-2 (2001)

[7] Byrne, E.J., A Conceptual Foundation for Software Re-engineering, in Conference on Software Maintenance 1992.

[8] Tahvildari, L., Kontogiannis, K. On the role of design patterns in quality-driven re-engineering. in Proceedings of the IEEE 6th European Conference on Software Maintenance and Re-engineering (CSMR). 2002. Hungary.

[9] G. Ar´evalo, S.D., and O. Nierstrasz. Discovering unanticipated dependency schemas in class hierarchies. in In Proceedings of CSMR '05 (9th European Conference on Software Maintenance and Re-engineering). 2005. IEEE Computer Society Press.

[10] Tahvildari, L., Kontogiannis, K., Mylopoulos,. Requirements-driven software re-engineering. in Proceedings of the IEEE 8th International Working Conference on Reverse Engineering (WCRE). 2001. Germany.

[11] S. Ducasse, T.G.ı., and J.-M. Favre, Modeling software evolution by treating history as a first class entity, in on Software Evolution Through Transformation 2004. p. 71–82.

[12] S. Ducasse, M.L., and R. Bertuli. High-level polymetric views of condensed run-time information. in Proceedings of CSMR 2004 (Conference on Software Maintenance and Re-engineering). 2004.

[13] S. Tichelaar, S.D., S. Demeyer, and O. Nierstrasz. A mmodel for Language-Independent Refactoring. in InProceedings of ISPSE '00 (International Conference on Software Evolution). IEEE Computer Society.

[14] Chia-chu chang "Software stability in software re-engineering". Information Reuse and Integration, 2007.

[15] Moghaddas, Y., & Rashidi, H. (2009). A novel approach for replacing legacy systems. Journal of Applied Sciences, 9(22), 4086–4090.

[16] Jakub Miler, Janusz Górski. "Identifying Software Project Risks with the Process Model", ICSSEA2004.

[17] Moghaddas, Y., & Rashidi, H. (2009). A novel approach for replacing legacy systems. Journal of Applied Sciences, 9(22), 4086–4090