Software Re-engineering: An Overview

Manar Majthoub, Mahmoud H. Qutqut and Yousra Odeh

Faculty of Information Technology

Applied Science Private University
Amman, 11931 Jordan
{manar.majthoub | qutqut | y_odeh } @asu.edu.jo

***Abstract*—The core of software re-engineering is to enhance or change existing software so it can be understood, managed, and reused as new software. When the system's software architecture and platforms become absolute and need to be changed, re-engineering is needed. The importance of software re-engineering lies in its ability to recover and reuse things which are already existing in an outdated system. This will obviously lower the cost of system maintenance and set up the basis for future software development. This paper presents brief overview of software re-engineering and highlights its future path.**

Keywords— software, software re-engineering, reverse engineering, forward engineering.

I. INTRODUCTION

The serious need of re-engineering emerged by the end of early 90s. This happened when users needed to shift their information systems into web-based interfaces [1]. Therefore, the field of re-engineering is born in order to solve the problems of old software systems that are essential for business operation, namely legacy systems. Since that time till present, researchers are working to design a process models and road maps that are flexible and repeatable in order to reengineer the legacy systems [2][4][3].

As beginning, it may be required to recall the definition of engineering in order to understand the re-engineering one. According to Sommerville, software engineering is defined as "engineering discipline that is concerned with all aspects software production from the early stages of system specification through to maintaining the system after gone to its use". Where re-engineering is concerned with re-implementing the legacy systems in order make them more maintainable. This requires a revisit to the old documentations in order to do corresponding re-documentation, revisiting the system architecture in order to reorganize the structure and re-implement it in modern technology or language that meets the present and market needs [2]. All this should be done taking into account maintaining the major functionality of the re-engineered system. The difference between engineering and re-engineering is that the former starts with the specification and derives the target system through design and implementation. Therefore, it is called forward engineering. Whereas re-engineering starts with the existing system as its specification and derives the target system through understanding and transformation [2]. In short, re-engineering is based on the concept of reuse from legacy systems. Software re-engineering is needed in several cases. The following are some cases where it is needed.

- When the code no longer has a clear and logical structure, and documentation may not exist.
- When hardware and software support in current systems become outdated and obsolete due to changes in organization policies, market competitive or survival needs.
- When the developers of the legacy systems are not available to verify or explain information, and the only source is the current software code
- When Legacy systems through years of modifications become difficult or risky and expensive to change.

This paper presents an overview of software re-engineering for the readers and ends by showing the anticipated future path of the software re-engineering. The paper is structured as follows. Section II presents the general model of software re-engineering. Different re-engineering approaches are presented in Section III. Section IV follows with presenting enhanced approach of software re-reengineering. Finally, we highlight some future paths of the software reengineering and concludes the work in Sections V and VI.

II. TRADITIONAL SOFTWARE RE-ENGINEERING MODEL OVERVIEW

Chikofsky and Cross were the earliest researchers who defined software re-engineering as "the examination and alteration of software systems to reconstitute it in new form and the subsequent implementation of the new form" [6]. The method of software reengineering usually includes a set of other set methods, namely, forward and reverse engineering, re-documentation, restructuring and translation. [4]. The objective of software reengineering is shown in Figure 1 through understanding the existing software or its functionality, redesigning and improving it with the latest technology [2]. This figure shows the difference in the inputs and outputs for both software engineering and its re-engineering.

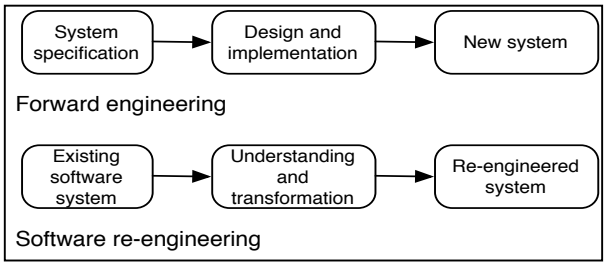

Figure 1. The objective of software engineering and re-engineering (reproduced from [2]).

Re-engineering starts with the source code of the current systems (i.e., as-is systems) and ends with the source code of the new system (i.e., to-be system). This process can be too simple, as when we use the translation tool to translate code from one language into another (C to JAVA), or from one operating system to another (Windows to UNIX) [8]. However, it can be very complex when using the source code in order to redesign and determine the requirements in legacy systems and compare them to new system requirements taking in to account removing the things that are no longer needed [8].

Figure 2 shows the traditional process of re-engineering software within all its regarding software development level abstractions [8]. Three key principles are applied in this model shown in Figure 2 that are abstraction, alteration, and refinement. The abstraction level is the gradual increase of system information abstraction level. The more abstract the information the more abstract the characteristics that describe the system [8]. In Figure 2, this principle moves upward and this is called the reverse engineering with its associated sub-processes, tools and techniques. The alteration principle refers to the conduction of one or more changes in system abstraction without changing the level. This involves extension, deletion or modification with respect to functionality [8]. Finally, refinements can be described as the opposite principle of abstraction. It refers to the gradual decrease of abstraction level regarding system information representation by replacing it one with more detailed information. The model shows that an alteration of an existing code to another language does not require a reverse engineering. However, if it is required to re-specify requirements, reverse engineering must be done at all abstraction level starting from implementation to design in order to have the functional characteristics [8].
It is very crucial to follow some tasks while reengineering as below [8].

- Forming re-engineering team
- Analyzing project feasibility
- Analyzing and planning
- Implementing the reengineering
- Testing and transition

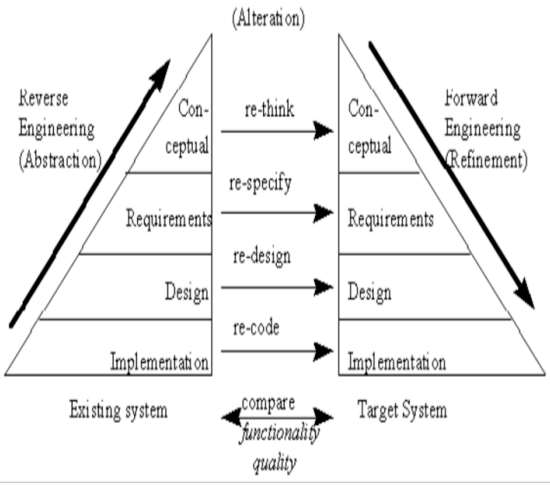

Figure 2. The traditional model of re-engineering software (adapted from [8]).

A. *Reverse engineering*

Reverse engineering is "the process of analyzing a subject system to identify the system's components and their interrelationships and create representations of the system in another form or at a higher level of abstraction" [8]. The definition shows that reverse engineering does not generate new information to the system. The objectives of reverse engineering are to retrieve lost/undetermined information, detect side effects and apply the reuse [8]. The reverse engineering process is shown in Figure 3.

Reverse engineering often comes before re-engineering. Usually, the reverse engineering is preferred When the specification and design of the current system must be determined before using them as input for the specification of requirements for replacement systems especially if the specification does not exist.

B. *Forward engineering*

By moving downward through the abstract level, new system can be created. Forward engineering can be described from the opposite side of reverse engineering. In another word, forward engineering is the traditional software development where information is elaborated and described in more detail while moving from conceptual design, to detailed design, to physical implantation. This forward direction of engineering puts the project at high risk for any changed requirements [8].

III. SOFTWARE RE-ENGINERRING APPROACHES

This section presents the three different approaches of software reengineering. The approaches differ in quantity and rate of replacement of legacy systems to the new system. Each approach has benefits and risks.

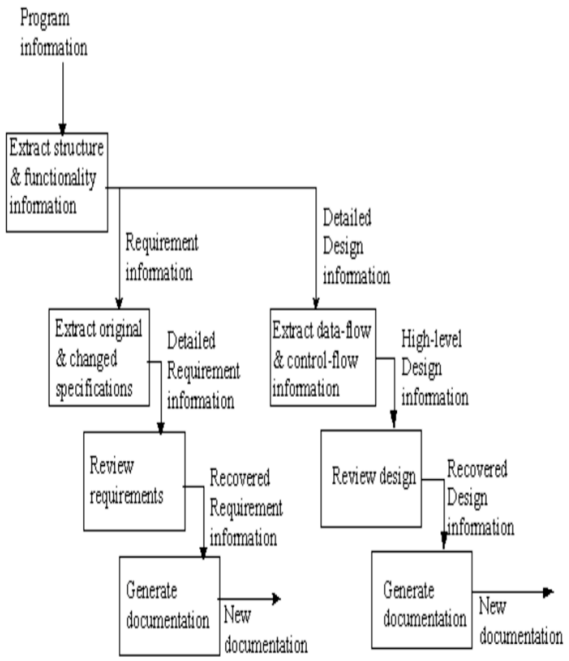

Figure 3. The process of reverse engineering (adapted from [8]).

A. *Big Bang Approach*

This approach is known as "Lump Sum"; where at one time, overall system is replaced. This approach is used when project needs a problem solving immediately. For example, migrating the current system to different architecture [8]. Figure 4 shows this approach. In short, the advantages for this approach is allows the system to run in new environment without any required integration of interfaces. A disadvantage of this approach that it is not suitable for large projects that consume a lot of time and resource before the target system is in place [8].

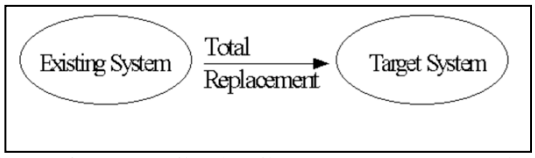

Figure 4. The BIG BANG approach (adapted from [8])

B. *Incremantal Approuch*

This approach is known as "Phase-out" or "Additive" as shown in Figure 5. In this approach, each part of the system is re-engineered and added in the form of new version in order to meet the ultimate needs. This approach has advantages where parts of the system are produced faster with simplicity in tracking errors. Therefore, it lower in risk than Big Bang approach. Also, customers are satisfied through gradual experience of the new quickly delivered each one part of the target system. However, the complete delivery of all parts will take long time and this is considered as disadvantage. Another disadvantage is that the overall structure of the system is not possible for alteration except parts that are re-engineered [8].

C. *Evolutionary approach*

This approach is similar to the incremental approach where each component of the old system is re-engineered and replaced by new one in the target system, as shown in Figure 6. However, the replacement is conducted based on functionality not on the structure of the old system. Here, developers focus on creating cohesive components.

Advantages of this approach is the system components are described as highly cohesive. Therefore, this approach is suitable for converting current system into object-oriented. Some interface and response time problems may occur. Another disadvantage is that similar functions must be refined as single functional unit in the new system.

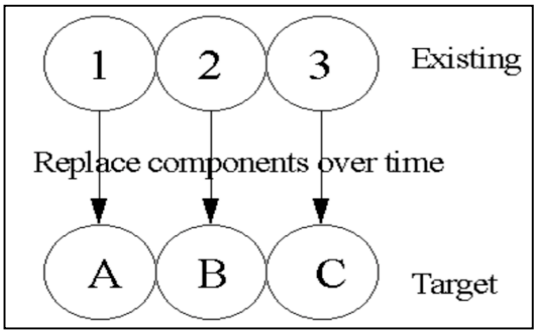

Figure 5. Incremental Approach (adapted from [8])

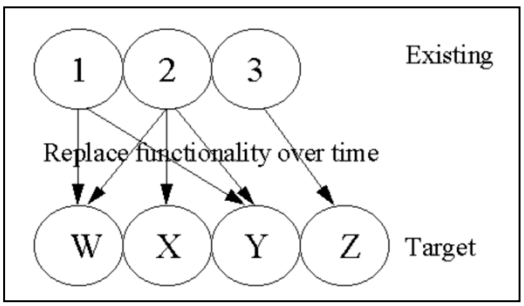

Figure 6. Evolutionary approach (adapted from [8])

IV. ENHANCED RE-ENGINEERING MECHANISM

Enhanced re-engineering mechanism is a software re-engineering process which takes advantage of multiple abstraction methods and levels, to transform an existing software system to a new target software system. Figure.7 shows the process of enhanced software re-engineering mechanism [5]. Here, the system uses both forward and reverse engineering techniques. At first, this mechanism performs the feasibility study to test the compatibility of the system, and then construct the components required for that process. Then after gathering the requirements, it goes to the second phase that is mapping of Restructured Software Requirements Specification (SRS) in order to finish the design document. The re-designed document is an output of the second phase. In third phase, the programming part is being customized based on the changes that have been done in the re-design document phase. It can able to return over from this phase to second phase and vice versa. Then this mechanism performs re-testing and re-integration of existing software modules to perform a specific function. In this phase, the system compares the performance of existing software with the new software. As per the result, the better algorithm is exchanged with the existing system. The advantages of this process that it reduces the complexity and improves the quality of new re-engineering software. After completing the integration of different units, the modified system should be implemented to get the target system which is required by the user [5].

The working methodology is as follows and explained in next sub sections. Enhanced re-engineering mechanism has five phases which they are feasibility study and requirements, restructured SRS, design to code, Comparison of Existing and proposed Functionalities and Implementation [5].

A. *Feasibility study and requirements*

In this work, the initial stage is the feasibility study and requirements. In this stage, the feasibility study is done to check the configuration and compatibility of the computer system. After finishing the feasibility study, the needs of the system are re-specified based on the user's desire. The SRS has all requirements in a written structure as authorized document. To re-specify the system requirements, the system needs to map this with the SRS [5].

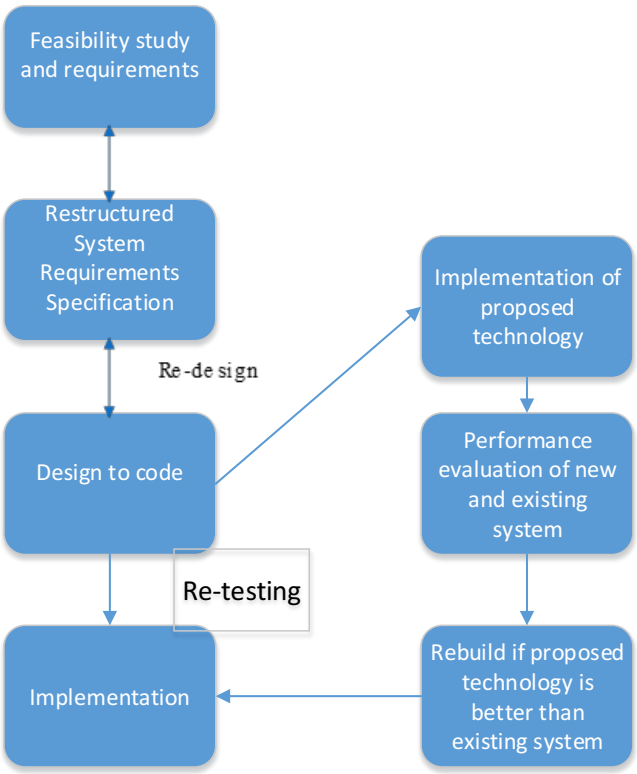

Figure 7: Enhanced re-engineering mechanism process (adapted from [5]).

B. Restructured system requirements specification

This stage portrays in detail the restructured SRS Process. Documentation is an important attribute in the software development process because it reproduces the components of the total re-engineering process and performs as a planner for the end product. Here, the experts make a comparison between the requirements of the existing system and with the new mechanism. SRS is used to integrate both the new SRS with existing SRS [5].

C. Design to code

This stage offers the details about design to code process. In this stage, according to the re-design document code is done by the programmer. Normally, old algorithms are implemented in traditional development languages and with existing functionalities which are required to be re-written. For example, if a system technology required the necessary of time and accuracy and has become old to be accurate and need new algorithms so the system, then it should be planned to re-engineering with a new technique [5].

D. Comparison of xxisting and proposed functionalities

This stage gives the details about re-testing process. For re-testing, firstly both existing and new software application are taken. Then, it compares the performance of functionalities of the existing software application with the functionalities of the new software application. For evaluation, if a system uses the standards like memory usage, running time and system configuration. Then, the old function performance is compared with the proposed algorithm. According to the outcomes, rebuilding process should be performed. In comparison result, if an algorithm gets more performance than other, then the better performance algorithm is replaced for re-build process [5].

E. Implementation

This stage is the last stage of enhanced re-engineering mechanism. According to the result of previous re-engineering stages, the implementation of a software application can be complete. In the implementation, a particular part can be replaced with the good one which fully relies on the previous four stages of the enhanced software re-engineering mechanism [5].

V. FUTURE OF SOFTWARE RE-ENGINEERING

Due that software re-engineering begins with the code at legacy systems, then code understanding requires attention as it is the only complete source of information especially if the system documentation is poorly specified or outdated. The authors predict the future of software re-engineering as below.

First, software companies should generate wide range of intelligent re- and reverse engineering tools that would be able to detect pieces of code as components and identify their responsibilities. All these components can be viewed in catalogue for many benefits. By viewing and scrolling between components, the team can search for a component and detects missing if any. This catalogue may help in showing how many times a component has been reused in the overall system. Hence, this highlights the criticality of this component and thus requires extra attention while re-engineering. Also, the catalogue shall show the relations between components and dependency levels of other components on a particular one. Reengineering is an improvement of an already existing system. It's not a new system capable of solving all problems.

Second, the tool shall be able to generate a code driven architecture and diagrams based on selected components from the catalogue. The generated architecture or diagram shall highlight the weakness points/era based on programmed fundamental design principles in order to guide the designers and developers in reconstructing the system. More intelligent tools may advice the designers regarding the most appropriate design. For example, assume a component is accidentally designed as highly coupled, then the reverse engineering tool would show a suggestion to create another component with redesigned responsibility in order to reduce the coupling risk.

Third, the power of re-engineering relies on the value of information derived from legacy systems and its continues understanding. This information of this system should address the needs of each role in the reverse engineering team starting from manager, business analyst, designer,

developer, tester, quality assurance, etc. For example, manager shall be able to use the tool in order to assign tasks and schedule the project plan. Developers shall be able to view the impact of code changing. Therefore, forward and backward traceability of changes in reverse engineering shall be supplied in order to reduce risks of required changes [7].

Fourth, one of the challenges is to show the knowledge of non-functional requirements. This is due to the absence of diagrams that illustrate this type of requirements during traditional development process. The research community shall pay extra attention for considering the design of non-functional requirements with similar attention of designing system functionalities during forward engineering. [9]. This is in order to conduct the appropriate reverse engineering of non-functional requirements.

Fifth, another challenge is to ensure that the applied re-engineering shall not contradict the ultimate business goals of the organisation. This challenge is similar the aforementioned one (i.e., challenge of showing knowledge of non-functional requirements). At present, designers still do not embed the orientation of business goals in the system diagrams due that all attention in practice is given to system functionalities or services from both technical team and business team. The area of modelling business goals is still evolving in the domain.

Sixth, the current gap between business process management and re-engineering is anticipated to be reduced. This is because of the evolved maturity in practice although with gradual pace regarding involving business process models in SRS.

Seventh, the authors anticipate an integration between the field of big data and software re-engineering. Big data technology helps in extracting the information of user experience and satisfaction levels of a system that would to path the way for identifying candidate potential changes in order to continuously improve user satisfaction.

The more consistent and repeatable re-engineering process and the more powerful the tool used in providing the required information of changing business rules in reverse engineering, the more maintainable the targeted system. The science of re-engineering, if widely practiced, is anticipated to show the timely growth of business for an organisation where each time a system is re-engineered in order to address a need. Designing and constructing code is an art. This is said as the same with redesigning and reconstructing the code, that is the art of re-engineering.

I. CONCLUSION

At present many structure designs methodologies have been developed. This resulted in improvement of software reliability and maintainability and decreased maintenance time. However, most companies have legacy systems that are out of date which they don't replace with new ones. This is because such outdated systems contain corporate information and decisions that they don't wish to be lost when replaced with a new one. Also, old legacy systems are an investment which costs a lot to develop and evolve just to be easily discarded. For these above-mentioned reasons, re-engineering becomes a useful tool to convert old obsolete systems to more efficient ones. Re-engineering is a structured discipline of software development. However, extensive work is still needed. This is because there is no methodology for evaluating the quality of the newly restructured system.

The absence of tools is one of the reasons that organizations neglect or conduct a not real practicing of re-engineering. Also, the absence of a skilled team specialized for conduct re-engineering is still challenging in organizations that wish upgrading their legacy systems. The availability of such would help the team in re-engineering task. The authors concluded with some functions to be designed in the future re-engineering tools.

ACKNOWLEDGMENT

This work made possible by a financial support from Applied Science Private University in Amman, Jordan.

REFERENCES

[1] H.Muller, J.Jahnke, D. Smith, M.A Storey, S. Tilley and K.Wong, "Reverse Engineering: A Road Map, ICSE'00 Proceedings of the Conference on The Future of Software Engineering, pp. 47-60, 4-11 June 2000.

[2] I . Sommerville, Software Engineering, 9th edition, Pearson Publication, London, 2014.

[3] A. Abbas and W. Jeberson, Proposed Software Re-Engineering Process That Combine Traditional Software Reengineering Process With Spiral Model, International Journal of Advanced Computer Research vol.4, pp.75-83, January 2013.

[4] F. Weil, Legacy Software Reengineering, Unique Soft LLC, 2015.

[5] D. Chidambaram, Enhanced Re-Engineering Mechnanism to Improve the Efficiency of Software Re-Engineering, International Journal of Advanced Computer Science and Applications, vol.7, pp. 285-290, 2016.

[6] E. Chikofsky, Elliot and, J. Cross, Reverse Engineering and Design Recovery: A Taxonomy", IEEE Software, Vol 7, No 1, pp. 13-17, January 1990.

[7] B. Y. Alkazemi, A Framework To Assess Legacy Software Systems, Journal Of Software, Vol. 9, pp. 111-115, January 2014.

[8] D. L. H. Rosenberg, "Software Re-engineering, Goddard Space Flight Center, NASA.

[9] L. Chung., B. Nixon and J. Mylopoulos, Non- Functional Requirements in Software Engineering. Kluwer Academic Publishers, UK: London 2000.