EXPLORE iJET International Journal of Engineering & Technology
CENTRE OF PROFESSIONAL RESEARCH PUBLICATIONS
ISSN 2049-3444

International Journal of Engineering and Technology Volume 2 No. 2, February, 2012

The Need of Re-engineering in Software Engineering

Ahmed Saleem Abbas, W. Jeberson, V.V. Klinsega
Department of Computer Science & Information Technology, SHIATS University, India

ABSTRACT

In this paper we will discuss the importance of software re-engineering and the reasons behind this importance followed by a discussion on each of these reasons with examples to prove that the re-engineering process is a useful tool to convert old, obsolete systems to more efficient, streamlined systems. And the re-engineering is used to increase maintainability, interoperability, performance and testability. Also re-engineering is used to decrease personal dependency.

Key words: *Software Engineering, Software Re-engineering, restructuring, legacy systems*

1. INTRODUCTION

Software Re-engineering 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. The process typically encompasses a combination of other processes such as reverse engineering, re-documentation, restructuring, translation, and forward engineering. The goal is to understand the existing software (specification, design, implementation) and then to re-implement it to improve the system's functionality, performance or implementation [1].

There are four re-engineering objectives, they are: Preparation for functional enhancement, Improve maintainability, Migration, and Improve reliability [1]. Figure 1 refers to the general model of software re-engineering and illustrates the reverse engineering and forwarded engineering.

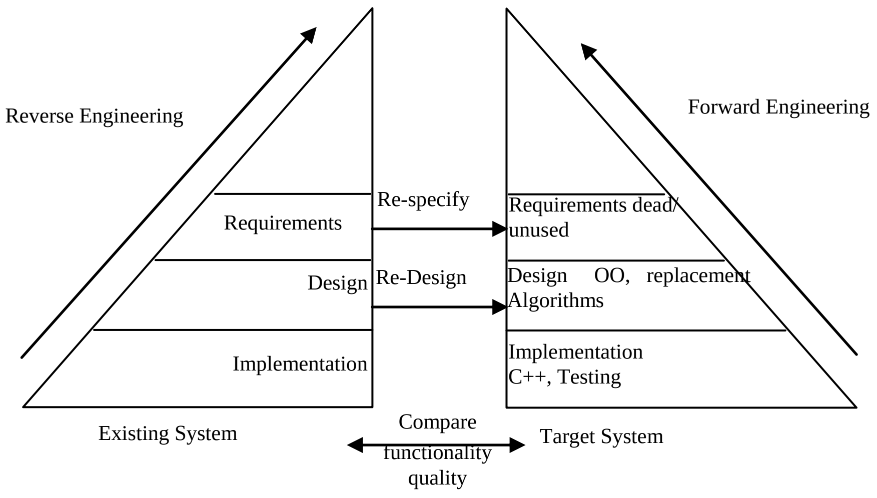

Figure 1. General Model for Software Re-engineering

For examples when one would wish to re-engineer code could be the modification of a date field (Y2K) or possibly re-modularization of code to facilitate later maintenance. In a typical Y2K scenario, one would expect to find transformation of items such as the size of the date field in a record, and other transformations that are concerned with code that is used for outputting dates.

2. WHY SOFTWARE RE-ENGINEERING?

The need of software re-engineering evolved from the following:

- Increasing legacy software.
- Emerging technologies.

- Decreasing ratio of Successful projects.
- Increasing number of competing Companies.
- More demand for quality attributes.
- Changing attitude of people.
- Insistence for Software maintenance.

3. THE BENEFITS OF SOFTWARE RE-ENGINEERING

The main benefits of software re-engineering are as follows:

- Increased maintainability [2].
- Improved performance [3].
- Increased interoperability [4].
- Decreased personal dependency [5].
- Improved testability [5].

3.1 Increased Maintainability

Many resources are spent on software maintenance. Thus, producing software that is easy to maintain may potentially save large costs. The problem of maintaining software is widely acknowledged in industry, and much has been written on how maintainability can be facilitated by e.g. tools and processes. However, you cannot control what you cannot measure, and there is yet no universal measure of maintainability. Some proposals have indeed been presented, but the very idea of measuring maintainability has inherent problems [6].

Maintainability: The ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment [7].

In practice, when measuring maintainability of a system during a long sequence of changes, if it is true that the system decays continuously, the resulted graph will be similar to the one in figure 2 [6].

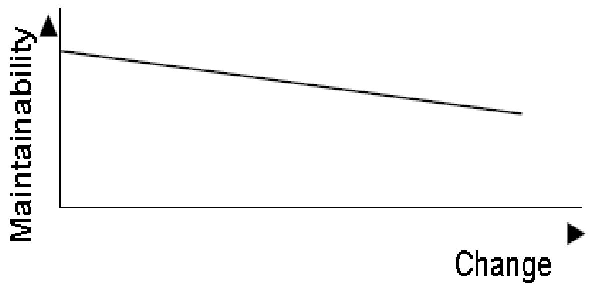

Figure 2. the system deteriorates (measured in maintainability) as changes are implemented [6].

The maintainability sometimes increases and sometimes decreases, that was expected. For example, the logical change “system restructuring” should cause the maintainability measure to increase, as Figure 3 describes [6].

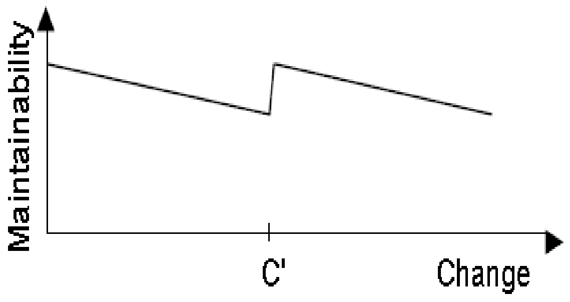

Figure 3: Change C’ (a restructuring of the system) causes maintainability to (temporarily) increase [6].

As was known, the software re-engineering has many activities and the restructuring activity is one of them so the maintainability will increase when the SRE is applied.

3.2 Improved Performance

Performance (responsiveness and scalability) is a make-or-break quality for software. Poor performance costs the software industry millions of dollars annually in lost revenue, decreased productivity, increased development and hardware costs, and damaged customer relations. When performance problems occur, they must be fixed immediately. In response, the project often goes into “crisis mode” in an attempt to tune or even redesign the software to meet performance objectives. In these situations, it is vital to maximize the performance and capacity payoff of your tuning efforts [8].

There is a systematic, quantitative approach to performance tuning that helps you quickly find problems, identify potential solutions, and prioritize your efforts to achieve the greatest improvements with the least effort. The steps are:

- Figure out where you need to be.
- Determine where you are now.
- Decide whether you can achieve your objectives.
- Develop a plan for achieving your objectives.
- Conduct an economic analysis of the project.

Using this approach has been shown to provide a high payoff for a relatively small cost. Once you run into trouble, tuning the software is likely to be your only choice. However, it’s important to realize that a tuned system will rarely, if ever, exhibit the level of performance that you could have achieved by re-engineering that system. The key to achieving optimum performance is to adopt a proactive approach to performance management that anticipates potential performance problems and includes techniques for identifying and responding to those problems early in the process. With a proactive approach, you produce software that meets performance objectives and is delivered on time and within budget, and avoid the project crisis brought about by the need for tuning at the end of the project. Software performance engineering (SPE) provides a systematic, quantitative approach to proactively managing software performance [8].

Figure 4 illustrate how the software re-engineering process increases the performance of the system [9].

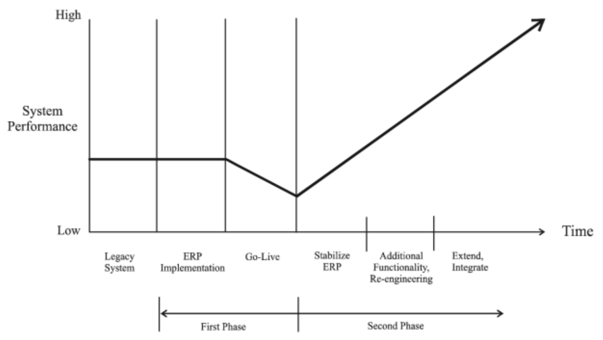

Figure 4. The effect of SRP on system performance [9].

3.3 Increased Interoperability

The IEEE Standard Glossary of Software Engineering Terminology defines interoperability as “the ability of two or more systems or components to exchange information and to use the information that has been exchanged [10].

The Department of Defense Directive defines interoperability as “the ability of systems, units or forces to provide data, information, materiel, and services, to and accept the same from other systems, units, or forces and to use the data, information, materiel, and services so exchanged to enable them to operate effectively together.” Figure 5 refer to the Levels of Information Systems Interoperability (LISI) [10].

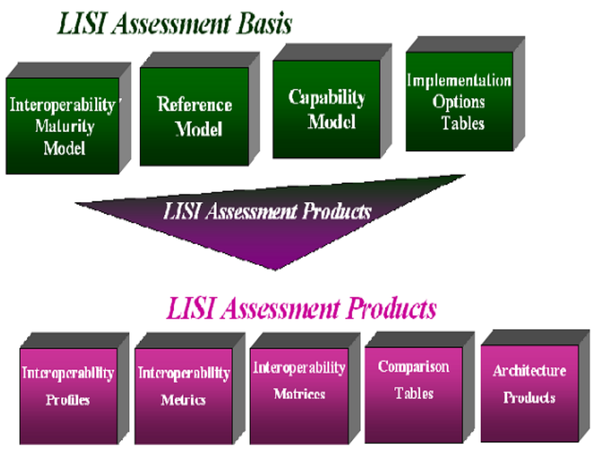

Figure 5. A Process for Interoperability [10]

System interoperability involves the efficiency of producing and refining software to interact with new and existing hardware to reliably exchange information between systems. More specifically, system interoperability encompasses the idea of software process and software modeling involving legacy systems and new hardware/software that must interact to provide reliable exchange of information between systems. Dynamic systems engineering requires that both processes and models be flexible to allow re-engineering and reevaluation of the processes and models themselves to improve the interoperability [10].

3.4 Decreased Personal Dependency

Large-scale software development requires coordination within and between very large engineering teams which may be located in different buildings, on different company campuses, and in different time zones.

Coordination between software development teams is one of the most difficult-to-improve aspects of software engineering. Kraut and Streeter argue that the software industry has been in crisis mode for its entire existence, and a root cause is the difficulty in coordinating work between teams of developers. Researchers have studied professional software development teams empirically to gain greater understanding of how software development processes, tools, and people impact coordination. The importance of intra- and inter-team coordination is a foremost concern as software development increasingly becomes globally distributed, and remains a persistent challenge in other disciplines as well.

Coordination is a decision-making that required communication, capacity and cooperation. These three components of coordination are necessary, but by themselves insufficient, for coordination to take place. Communication is necessary because person A needs to communicate to person B, in some form, what needs to be done, and B needs to understand the communication. Capacity is necessary because B needs to be able to do what is required of him. Cooperation is necessary because B needs to be willing to do what is required of him. If any of the three necessary components are lacking, the outcome will be less than ideal [11].

From that, the personal dependency is important and take a lot of effort to coordinate the software development team work, so that, Software re-engineering is used to decrease personal dependency by restructuring the legacy system in a manner that reduce the dependency of person A on person B where (both A & B) are members in the development team [5].

3.5 Improved Testability

Test costs are driven by the size and complexity of the software. Size can be measured in terms of the number of elements making up the system. These could be statements, methods, classes, components, interfaces, files, database tables and GUIs. Complexity is measured in terms of the number of interactions between the elements. These could be associations, calls, messages, file transfers, database accesses, import, exports and events from outside. The less there are, the less there is to test [5].

Another factor which influences testability is the visibility of the data interfaces. Data passed between components can be encoded in internal data formats or it can be passed as readable character strings [5].

A very critical factor in testability is the separation of the user interface from the processing logic. This separation of presentation from processing is a prerequisite to testing the processing, i.e. the business logic, without having to enter the data in the user interface, which requires a lot of time and is difficult to automate. A final factor in reducing test effort is the separation of the data access operations from the data processing [5].

The goal of re-engineering for testability is to restructure the software in such a way that testability criteria can be met while at the same time reducing the size and the complexity of the system. Re-engineering software for testability is definitely a worthwhile effort. Identifying and removing clones, refactoring deeply nested code and restructuring the architecture are tasks that can be automated. Several tools exist which support that. By using them, re-engineering costs can be minimized. Other tasks such as algorithm optimization, merging data accesses and simplifying user interfaces can be done manually at a rather low cost. In view of the potential savings in testing costs, it is well worth it to invest in a re-engineering project running parallel to the development project. Re-engineering for testability can still be worthwhile before going into system testing. It might also mean that the development process is being complemented by a parallel re-engineering process, intended to raise the quality of the software, including testability. Figure 6 refer to parallel projects [5].

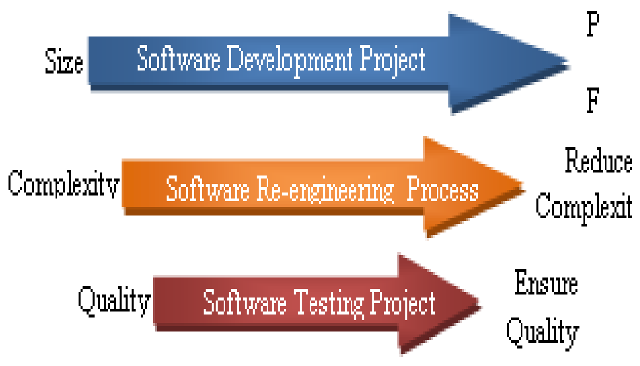

Figure 6. Parallel Projects

4. CONCLUSION

Many modern software design methods have been developed to improve the reusability and maintainability of software and to reduce the time required for the maintenance and development operations, but many companies have old or legacy software systems, these companies spend a lot of money to maintain their old systems. These systems cannot be replaced by new systems because they contain implicit information and decisions cannot be lost. For these purposes, re-engineering becomes a useful tool to convert old, obsolete systems to more efficient, streamlined systems.

REFRENCES

[1] Dr. Linda H. Rosenberg, "Software Re-engineering" Software Assurance Technology Center, p. 2-3 Linda.Rosenberg@gsfc.nasa.gov

[2] Fowler, M.: Refactoring–improving the design of existing code, Addison-Wesley, Reading, MA., 1999, p. 53

[3] Sneed, H.: “Measuring the Reusability of Legacy Software Systems“, Software Process – Improvement and Practice, Wiley Pub., No. 4, March, 1998, p. 43

[4] Sneed, H., Nyary, E.: „“Downsizing large Application Programs“, Journal of Software Maintenance, Vol. 6, No. 5, Oct. 1994, p. 235-248

[5] Harry M. Sneed, Anecon GmbH, Wien, " Re-engineering for Testability", Universities of Regensburg and Passau, May, 2006

[6] Rikard Land, "Measurements of Software Maintainability", Department of Computer Engineering, Mälardalen University.

[7] IEEE, IEEE Standard Glossary of Software Engineering Terminology, report IEEE Std 610.12-1990, IEEE, 1990.

[8] Lloyd G. Williams, Ph.D, Connie U. Smith, Ph.D. "Five Steps to Solving Software Performance Problems", June, 2002.

[9] T. Hillman Willis, Ann Hillary Willis-Brown, (2002) "Extending the value of ERP", Industrial Management & Data Systems, Vol. 102 Iss: 1, pp.35 – 38

[10] Cadet Pamela A. Sanders, Dr. John A. Hamilton, "A Process for Interoperability", Auburn University.

[11] Christopher Poile, Andrew Begel, Nachiappan Nagappan, and Lucas Layman " Coordination in Large-Scale Software Development: Helpful and Unhelpful Behaviors", Dept. of Computer Science, North Carolina State University, September 28, 2009.