A QUALITY-ORIENTED APPROACH TO SOFTWARE REENGINEERING

Murat Pasa Uysal, Turkish Military Academy, Learning and Research Center, Ankara, Turkey.
+90(312) 417-5190, x4440, mpuysal@kho.edu.tr
A. Erhan Mergen, Rochester Institute of Technology, Saunders College of Business
Decision Sciences, 107 Lomb Memorial Drive, Rochester, N.Y. 14623-5608.
(585) 475-6143, emergen@saunders.rit.edu

ABSTRACT

Legacy software systems can be regarded as the products with defects, which do not conform to quality standards. Their mission-critical functions make them valuable for organizations. Software reengineering (SR) has been used as a primary method for the improvement of these systems. However, the current SR models or solutions are thought to have different types of deficiencies. The purpose of this paper is to present a quality-oriented SR methodology. We integrate software engineering activities, the Six Sigma process, and the Software Quality Requirements and Evaluation (SQuaRE) standards. We believe that this model would have great value for customers, software professionals, or anyone who is responsible for defining and evaluating the quality of a software product to be reengineered.

Key words: Software reengineering, six sigma methodology, SQuaRE standards.

INTRODUCTION

A great majority of software in the industry has been built over the past decades and they are now considered “legacy” systems. Even though the age of software could be the major determinant for being a “legacy system,” a relatively new software system could be considered legacy due to its poor maintenance. As time passes, keeping these systems up-to-date becomes complex and time-consuming [2]. Their source code may not be comprehensible and it may no longer be an easy job to maintain these software systems [1] [13]. However, discarding or redeveloping these systems has many risks [15] [17]. Their mission-critical functions, and possibly undocumented business and programming logic make them valuable for organizations. This type of system generally represents the accumulated knowledge, experience, or familiarity of technical teams, users and managers. Software reengineering (SR) allows for the analysis, design and improvement of these systems.

PROBLEM

SR is a complex process. It intrinsically aims to improve the quality of a legacy, as well as reduce performance variations resulting from poor quality. A legacy system can be regarded as a software product with defects and non-conforming to quality standards, which leads to unsatisfied users [18]. SR could be one of the primary options for improvement. The literature
review and the case studies, however, indicate different types of deficiencies pertaining to current SR solutions, such as:

- The proposed solutions are problem-specific. They generally cannot provide generic quality improvement solutions [16].
- They are not applicable to all types of software, such as large-scale systems, enterprise information systems, embedded systems, off-the-shelf software products, etc. [3].
- The majority of these solutions look at a SR problem from the software engineer’s or developer’s point of view [4].
- These SR solutions cannot assure conformance to international software quality standards (ISO/IEC 25000n [7]).
- They cannot provide prescriptions leading to continuous software improvements after the delivery of a reengineered product.

**METHODOLOGY**

The purpose of this paper is to present a process model to improve the SR process. Through this model, we not only intend to improve process, but also present a quality-oriented method to SR. This aims to assure quality of any reengineered product by reducing its defects. SR in essence is a quality improvement process. We believe that any SR solution should have three mainstays: (1) a quality improvement philosophy, (2) an orientation to quality issues in any SR project, and (3) the assurance of software quality as confirmed against the standards. The novelty of this study is the integration of SR activities, Six Sigma, and the Software Quality Requirements and Evaluation (SQuaRE) standards to create a “quality-oriented SR model” (Figure 1). The Six Sigma program establishes the process framework for improvement; software engineering provides the tools and techniques for software process; and the SQuaRE standards includes the quality models, descriptions, and specifications required for a quality-oriented SR approach.

Although Six Sigma can be regarded as a single process, it is typically divided into five phases, referred as the “DMAIC”. This is an acronym for: “define”, “measure”, “analysis”, “improve” and “control” phases. Six Sigma encapsulates the quality-oriented process framework. A Six Sigma project is usually aimed towards achieving a “breakthrough improvement”, such as “reengineering” [5]. SR activities include software engineering tools, techniques, and practices. Finally, the SQuaRE provides the “software quality requirement” (ISO/IEC 2503n [10]), “software quality measurement” (ISO/IEC 2502n [9]), and “software quality evaluation” (ISO/IEC 2504n [11]) procedures, standards and metrics needed for our proposed model. It is an approved series of standards for software quality. It helps software developers, acquirers and evaluators by providing descriptions associated with requirement specification and the measurement and evaluation of software quality (ISO/IEC 2501n [8]). However, it is only devoted to the system and software product quality as a quality reference model. It naturally does not include a ready-to-implement process framework. Therefore, it has to be emphasized that the DMAIC process provided the framework needed for our quality-oriented SR approach.
Figure 1 depicts SR activities aligned with the DMAIC processes. The quality requirements of a SR process are expressed in the problem statements at the “define” phase. The “measure” phase identifies the current quality of the legacy software and its performance. The “analysis” phase helps SR teams to understand the nature of the problems and find their root causes. The reengineered version of the legacy product is produced through a series of software transformations and modifications at the “improvement” phase. The “control” phase includes both the control techniques and software maintenance activities. The arrows in Figure 1 represent the main process sequence and the data flows during a quality-oriented SR. The bidirectional arrows between the elements of the DMAIC process and SQuaRE indicate the references of SR teams to the SQuaRE standards at each step. This enables SR teams to define, measure, and control the software quality in terms of approved standards. Thus, a dynamic interaction is established amongst the customer requirements, the project requirements, and the software quality requirements through out a project lifecycle. The dotted arrows indicate that software improvements continue even after delivery of the reengineered product. The “replacement” phase indicates that a brand new system is designed after several improvements. We believe that this SR model would have a great value for customers and software professionals, as well as anyone who is responsible for defining and evaluating the quality of a reengineered product by providing a quality-oriented framework. The next sections present the phases of this quality-oriented approach to a SR process.
**Define:**
In this phase, a project charter is created and the mission statements, objectives, deliverables, and responsibilities are clearly stated. A quality-oriented SR approach requires not only a methodology, but also a philosophy change. The quality is viewed from the perspective of the customer to the systematic integration of people and the software process. Along with the assignment of a champion to the SR project, certain roles (master black belts, black belts and green belts) are described depending on the project scope and budget. In addition to the responsibilities the project charter also defines how the Six Sigma tools and techniques are applied and by whom. Important to this phase is defining software quality issues in the form of Six Sigma methodology. They are expressed as clear statements with measurable, attainable and problem-specific terms. It is worth noting that a Six Sigma project generally focuses on a single quality problem while a SR project may include several. Therefore, a number of Six Sigma SR teams can be established regarding the software quality problems. These teams solve software quality problems using the framework of ISO/IEC 2501n standards [8]. For example, using ISO/IEC 2501n standards, the “internal quality” is the degree to which static attributes of a legacy software product satisfy the stated and the implied needs (i.e., code complexity, number of faults, lines of code). The “external quality” is the degree to which this product enables the behavior of the system to satisfy the needs in a testing environment. The “quality in use” determines whether this product meets the requirements of specified users in a realistic environment. In these standards, the “quality in use” depends on the “external quality”, and the “external quality” depends on the “internal quality”. This interdependency naturally forms the quality lifecycle of our SR process model. Since customer-driven quality is fundamental to Six Sigma, our model puts emphasis on the customer needs and listens to the “voice of customers”.

**Measure:**
The main objectives of this phase are to gather information and to confirm, quantify and revise the problem. The focus is on how to measure the software product and the characteristics influencing its quality. It is vital to understand the causal relationships amongst software performance and the external, internal, and quality-in-use attributes of the product. When performing the measurements, appropriate software metrics, key input and output variables are determined as well as how these metrics would be tracked over time. The current software performance may need to be benchmarked.

For measuring the current software quality, we follow the guidelines provided by the ISO/IEC 2502n standards [9]. In these standards, the “quality measure elements” are defined as the base measurements used to obtain the “quality measures.” These elements may belong to internal quality characteristics, external quality characteristics, or quality-in-use characteristics. In other words, these characteristics measure the static representation of the software, behaviors of the software, and performance of the software respectively. Generally, it may not be practical to obtain all of the internal and external measures of a large legacy software system, and the “quality-in-use” measures within all possible user-task scenarios. Therefore, the ISO/IEC 2501n and 2502n quality standards should be tailored to the needs of a SR project. The priority should be given to customer requirements; it is fundamental to the Six Sigma philosophy.
Analysis:
This phase refers to the two-dimensional examination (process and software) of why the problems occur. The first dimension, which is the process analysis, relies on fact and data-based analysis of the business processes of the legacy software system, because we have to understand the nature of the problem and find the root cause without jumping to a quick solution. This is especially important for identifying improvement opportunities for the next phase. All work occurs in a system of interconnected processes and software modules within a legacy system. At this point, statistical techniques provide the means to draw accurate quantitative conclusions on business processes and software product. Six Sigma software teams try to identify potential root causes of a quality problem, and they try to validate the cause-effect relationships [15]. Since legacy software systems usually have poor documentation, we may need tools such as the “process maps” or “cause-effect diagrams” for the analysis procedures.

The second dimension, which is the software analysis, includes the software evaluation guided by the ISO/IEC 2504n standards [11]. This document describes the procedures for evaluating software quality from a broad view, for applicability to software developers, software acquirers, and software evaluators. It is also used for the analysis of different types of software, i.e., pre-developed software, commercial off-the-shelf software, and any custom software. This standard therefore complements our quality-oriented SR model by providing the evaluation procedures. The reengineered software product is categorized into static or dynamic. The static product involves the analysis of software design specifications and program source code. It is important to capture the design and architecture while identifying the relationships between the components of the system. The architecture and the functionalities are modeled to understand the rationale behind the legacy system. Often the static analysis may call for reverse engineering to extract information from the code. The dynamic analysis refers to the testing of the legacy software, both in a testing and operational environment. For both types of the software analysis, the primary objective is to identify the parts of the software responsible for the quality deficiencies and violations. These violations may belong to “internal quality”, “external quality” or “quality in use” measures of the legacy software. Thus, software analyses help SR teams detect error-prone code.

Improve:
In this phase, now that the root causes of the problems have been identified, the potential solutions are identified and implemented. The SR team(s) generates ideas for improving the software quality measures. The solution(s) may necessitate process and/or software product improvement. In either case, the chosen solution(s) should address the root causes. Furthermore, they have to be cost-effective and implemented within a reasonable time. It is also important to prevent a recurrence of the problem. The team(s) naturally gives the priority to the critical quality requirements and customer needs when eliminating defects and low quality. Process improvements can be accomplished in away similar to classical Six Sigma phases, and the software improvements are achieved using software engineering tools and techniques.

In terms of software reengineering, we can assume that requirement analysis, software modeling and software analysis procedures have been completed during the first three phases (define, measure and analysis) of our proposed model. The parts of the legacy software responsible for the poor quality have been identified. At the improvement phase, the customer requirements,
how to assure the quality standards, and the new business requirements need to be reviewed carefully before starting to improvement activities. It is worth noting that different types of requirements may have interdependent relationships and potentially conflicting goals [3]. However, all these requirements have to be satisfied with the new version of the legacy software. Therefore, it is important to specify a target software structure and software quality goals. This will guide the SR teams when eliminating the design and source code defects, while focusing on the target quality. An iterative and incremental software process model can be selected for applying software modifications and transformations [2] [14]. During the improvement phase, quality evaluation procedures are applied to each of the transformed parts of the code. This is done for determining whether the “internal quality” requirements, “external quality” requirements, and “quality-in-use” requirements are satisfactory met. This quality improvement cycle continues until the specified SR quality goals are achieved, and the targeted version of the legacy software is produced.

Control:
The control phase ensures that the improvements will be sustained by both tracking the software performance and carrying out proper maintenance activities. Controlling the product serves as the basis for an effective software management. This includes prescriptions and techniques pertaining to the period after delivery of the reengineered software product. Statistical methods and tools may greatly help perform the control activities. For example, Statistical Process Control (SPC) techniques, such as control charts, can be used to determine the stability of the product and process. SPC techniques have been shown to be effective when applied to software engineering [12]. A control chart can adopt different types of software quality indicators and control limits for monitoring the quality characteristics of a reengineered product. If one or more of these values fall outside the limits, an assignable cause of low quality is assumed to be present. Hence, collecting the measures and metrics during the phases of a quality-oriented SR project provides a strong insight and control over the evolution of software quality.

Controlling a reengineered product also means planned software maintenance. As we already know, in practice poor maintenance results in low quality software. Therefore, the control phase should also focus on software maintenance (Figure 2). IEEE defines software maintenance as “the process of modifying a software system or component after delivery to correct faults, improve performances or other attributes, or adapt to a changed environment” (IEEE 1219-1998 [6]).
IEEE puts software maintenance activities into four categories. “Corrective maintenance” deals with diagnosing and fixing errors in the source code after delivery. “Adaptive maintenance” aims modifying the system to cope with changes in the software environment. “Perfective maintenance” concerns functional enhancements to the system because of new or changed user requirements. “Emergency maintenance” is unscheduled corrective maintenance to keep a system operational. Based on these definitions, our quality-oriented SR model is capable of meeting these maintenance requirements. For example, our standard and measurement-based quantitative approach makes it easy to diagnose and fix errors during corrective maintenance. Its continuous improvements lead to software modifications required for adaptive maintenance. Since the focus is on customer satisfaction, a “perfective” maintenance is achieved. In Figure 2, the quality evolution of a reengineered product is given along with its possible versions. The phases, define, measure, analysis and improve, have already produced the reengineered product. Thus, the first version (V1) is obtained by the quality-oriented SR process at the time of t0. However, a software product usually has different types of requirements during its lifecycle. Therefore, the x-axis in Figure 2 represents the time and includes possible versions of the reengineered product in the future. As a result, these imaginary versions (from V2 to Vn) are assumed to be present at t2 and t_n after the delivery of the V1. This is, of course, achieved by the Six Sigma tools and techniques, as well as software maintenance activities.

CONCLUSION

Although there could be various technical and organizational reasons for replacing a legacy software system, there is a rationale for keeping a legacy system running as well. Over the past few years, organizations have been looking for the means by which they can improve these systems. SR is the primary method for improving quality while providing cost effective solutions and reduction in time and risk. There are various SR solutions in the literature. However, they generally focus on the development paradigms, and they are not applicable to all types of software. They also cannot provide prescriptions for the period after the delivery of a

Figure 2. The evolution of software quality through software maintenance
reengineered product. Furthermore, these models cannot assure conformance to international quality standards and satisfy all the stakeholders.

In this study, we proposed a quality-oriented SR method based on three important knowledge domains. The Six Sigma domain established the process framework and paved the way for the use of statistical methods for fact-based decisions. The software engineering domain provided the tools and techniques for a SR process. Finally, the SQuaRE standards domain included the quality models, descriptions and specifications needed for a quality-oriented method. We believe that our model can provide sustainable improvements while meeting the requirements of different types of people engaged in a SR project. A case study is planned in the future to see how this model can be applied to real-life situations. The study plan will naturally include a software product with defects in terms of quality and address how to improve the product quality through the proposed model. The implementation details and the results will be discussed in the next paper.

REFERENCES


