Software Requirements And Its Application In The Reengineering Process

Miguel Torres, Rayford B. Vaughn
Department of Computer Science, Mississippi State University
Mississippi State, MS 39762

ABSTRACT

The reengineering process is oriented to perform corrective or adaptive maintenance to a legacy system by changing the structure of the system to satisfy user’s requirements [5]. Usually the initial step to perform this task is made by a reverse engineering process, which extracts external documentation from the functioning system in a way that the reengineering process can be performed. The purpose of this paper is to present a technique based on the engineering requirements paradigm that can lead in dealing with the reengineering process and particularly in the reverse engineering step to increase its productivity.

Keywords: Reengineering, Reverse Engineering and Software Requirements.

1. INTRODUCTION / PURPOSE

The reengineering process can be viewed as the application of a major maintenance operation applied to a system (e.g. a software system). This process involves both reverse and forward engineering. In many cases no requirements, design or even development documentation for the legacy software exists. This lack of documentation leads to an unavailability of traces as well as making reengineering difficult and expensive [1].

Current trends in reverse engineering and reengineering process [1,2,3,5,6,7,8] usually assume that the user’s requirements for the system already exists, but during the process the traceability of those requirements is not evaluated or even worse, does not exist. That is why there is a gap in the reengineering process that can be filled by the use of the software requirements paradigm to identify specific reengineering goals.

2. REENGINEERING AND REVERSE ENGINEERING

The initial step in the reengineering process as stated by Lerner [5], is to find the source of the problem. This operation can be viewed as identification of the initial user’s requirements as stated in the requirements engineering process. According to Lerner [5], determining where the problem is physically allocated in the structure of the legacy software can be viewed as the process of identifying the derived requirements as stated in the requirements engineering paradigm. The success of this operation depends on how easily the maintainer of the application reads and understands the system and the size of it. Restructuring can be applied to small and medium size systems and can be made by the maintainer himself. This consists of performing corrective maintenance, which changes the system structure to make it more readable [3]. This approach will not be reviewed in this paper, since the maintenance is being performed directly in the structure of the software and its documentation and does not require a full understanding process to perform it. The other option - the one that this paper addresses is the reverse engineering process that extracts external documentation from a system to identify its components and their relationships for the purpose of maintenance or the creation of a new stand-alone system from the existing one used as a base [3,5].

Tahvildari and Kontogiannis [9] have proposed some steps for the reengineering approach. These steps are not well defined, and still need refinement since this area is still in development, and they are usually mixed with the forward engineering approach. The following are the reengineering steps proposed by Tahvildari and Kontogiannis [9]:

- Model Analysis – to understand the system design and architecture, provided by the design rules resulting from the reverse engineering process.
- Source code analysis – to understand the system’s implementation and the coding rules used to implement the system through the extracted components from the reverse engineering process.
- Remediation specification – to examine the particular problem and select the optimal transformation for the system.
- Transformation – to apply transformation rules in order to reengineer the system in a way that complies with specific quality criteria.
- Evaluation process – to assess whether the transformation has addressed the specific requirements.

Design patterns and its precursor the Unified Modeling Language (UML) has been adopted almost as the standard language for software design and Object Oriented Analysis and Design [4], and as supporters of the reuse of design elements, allowing designers and developers to communicate design information of a system in a more effective way. However, their primary use is still associated with the forward engineering and design face of the software life-cycle [8].

Some authors and in general the software engineering community are actually trying to create classifying schemes, to assist software maintainers in the generalization and improvement of a legacy system. Tahvildari and Kontogiannis [9] present some important definitions and schemes of pattern associations to enhance software quality during the reengineering process. Figure 1 extracted from [9] provides an overview on how concepts in the object oriented development and the reverse engineering paradigm are related. This data fusion model developed as a result of a research program on vulnerability correlation.
3. AIMS OF REVERSE ENGINEERING

The maintenance step of the software life cycle is usually one that developers give less attention. Reverse engineering processes aim at providing software maintenance but is not limited by it [3]. The most relevant aims of reverse engineering as stated by Klösch [3] are:

- Recovery of lost information – providing proper system documentation: Recovering information of old software design documents to support the maintenance activities. From the requirements engineering point of view this characteristic corresponds to the process of generation or annexing some documentation like the concept of operations document (ConOps).
- Assisting with maintenance – identification of side effects and anomalies: This characteristic provides additional information and leads to the refinement and discovery of the derived requirements that are to be stated in the software requirements specification document (SRS).
- Supports the improvement of software quality: The recovering of documents supports this assertion. This operation can also be achieved by the use of a requirement matrix that implements traceability of the new and old requirements of the system assigning them to the discovered architecture and design elements of the legacy system.
- Migration and software reuse: Sometimes the purpose of a maintenance operation is the migration of an application to new hardware/software. Since those changes are usually tied to each other, it is necessary to identify the structures (hardware/software) that will need to be changed and which structures will be useful for the development of future applications for the same user.

4. REQUIREMENT ENGINEERING SIMILARITIES

At this point we have discussed the characteristics of the reverse engineering process and how it is used as the main tenant of the reengineering process. Also, some common points and similarities were shown between the reverse engineering process and the requirements engineering paradigm.

In the next section we will summarize the information and similarities between the reverse engineering process and the requirement-engineering paradigm and provide a brief description of the requirements engineering process and how it fits into the reengineering process.

Requirements Elicitation

This first step in the requirements engineering process consists of finding the sources of the problem. In terms of reengineering, we talk about finding the origins of the problem and the new requirements of the system. This operation is supported by the documentation generated by the reverse engineering process and the ConOps document and by the use of interviews with the original development team (if possible) of the legacy system.

Requirements Analysis and Specification

The information collected in the previous step would help allow the maintenance team to find and assign each user’s requirement to a module in the original architecture of the legacy system and in identifying and generating the derived requirements of the system. This results in an SRS document and a matrix that will assign each requirement to an entity or unit in the legacy system, providing traceability of the requirements during the forward engineering process of the new system.
Requirements Verification
This step in the process of reengineering is very important in order to establish which requirements can be achieved and also verifies if the SRS is in compliance with the user’s requirements. It also can give guidance in how to focus the initial development efforts according to the technical limitations of the original system and the actual technology that can be applied in the maintenance of the legacy system.

Requirements Management
This step is the most important contribution of requirements engineering to the reengineering process since it brings to the reengineering process the order and equilibrium to keep track of the evolution of process and the activities associated with the solution of each requirement in the system.

5. CONCLUSIONS
We have show an overview of the reverse engineering and the reengineering processes and how the primary step of the reengineering process lies in the reverse engineering paradigm. We also show how the requirements engineering process is similar to the reverse engineering process and how using them together in the reengineering process can bring new tools and method to keep track and a more structured environment to perform the reengineering process. Some important characteristics that this proposed method brings to the reengineering process are the traceability of user’s requirements and the generation of additional documentation that assist the development process of the new system.

6. REFERENCES


