Chapter 1
Reverse Engineering and MDA: An Introduction

INTRODUCTION

Reverse Engineering is the process of analyzing available software artifacts such as requirements, design, architectures, code or byte code, with the objective of extracting information and providing high-level views on the underlying system.

A common idea in reverse engineering is to exploit the source code as the most reliable description both of the behavior of a software system and of the organization and its business rules. However, reverse engineering is immersed in a variety of tasks related to comprehending and modifying software such as re-documentation of programs and relational databases, recovering of architectures, recovering of alternative design views, recovering of design patterns, building traceability between code and designs, modernization of interfaces or extracting the source code or high level abstractions from byte code when the source code is not available.

Reverse engineering is hardly associated with modernization of legacy systems that were developed many years ago with technology that is now obsolete. These systems include software, hardware, business processes and organizational strategies and politics. Many of them remain in use after more than 20 years; they may be written for technology which is expensive to maintain and which may not be aligned with current organizational politics. Legacy systems resume key knowledge acquired over the life of an organization. Changes are motivated for multiple reasons, for instance the way in which we do business and create value. Important business rules are embedded in the software and may not be documented elsewhere. The way in which the legacy system operates is not explicit (Brodie and Stonebraker, 1995) (Sommerville, 2004).

On the one hand, there are billions upon billions of lines of legacy code in existence, which must be maintained with a high cost and, on the other hand, there is a high risk in replacing legacy systems.
that are still business-critical. The cost of reengineering should be significantly less than the cost of a new developing.

Reverse engineering does not involve changing the source legacy systems, but understanding them to help reengineering processes that are concerned with their re-implementing. Software reengineering starts from an existing implementation and requires an evaluation of every part of the system that could be transformed or implemented anew from scratch. This definition distinguishes the following main phases:

the examination and the alteration of a subject system to reconstitute it in a new form

and

the subsequent implementation in a new form. (Demeyer, Ducasse and Nierstrasz, 2002)

In other words, reengineering includes some form of reverse engineering followed by some form of forward engineering. Reverse engineering is the process of examination, not the process of change. Chikofsky and Cross (1990) define reverse engineering and forward engineering:

- **Reverse engineering**: “the process of analyzing a subject system to (i) identify the system’s components and their interrelationships and (ii) create representations of the system in another form or at a higher level of abstraction”

- **Forward engineering**: “the traditional process of moving from high-level abstractions and logical, implementation-independent designs to the physical implementation of a system”

Reverse engineering and related processes are using only three life-cycle phases: requirements specifications (including objectives, constraints and business rules), design of the solution and implementation (coding, testing, and delivery of the operational system) (Chikofsky & Cross, 1990).

Figure 1 depicts the relationship between tasks related to reverse engineering expressing transformations between or within abstraction levels linked to lifecycle phases. It also shows three processes related to reverse engineering: re-documentation, design recovery and restructuring.

Re-documentation is the creation of a representation of software artifacts that existed or should have existed within the same relative abstraction level. The resulting forms of representation are usually considered alternate views (for example dataflow, Abstract Syntax Tree, Patterns).

Design recovery rebuilds design abstractions from an integration of code, existing design documentation (if available), personal experience, and general knowledge about problem and application domains.

Restructuring is the transformation of a software artifact from one representation form to another at the same relative abstraction level, while preserving the artifact external behavior.

Reverse engineering has been used with two essential goals: design recovering and abstraction; however, it can be used to obtain more abstract representations with other purposes, for example testing, quality assurance, reuse and security.

Reverse engineering has been related with software evolution and maintenance. Software evolution is the process of initial development of a software artifact, followed by its maintenance. The ANSI/IEEE standard 729-1983 (Ansi/IEEE, 1884) defines software maintenance “as the modification of a software product after delivery to correct faults, to improve performance or other attributes, or to adapt
Related Content

Implementations: Discussing the Cases
www.igi-global.com/chapter/implementations/207088?camid=4v1a

Exceptions in Ontologies: A Theoretical Model for Deducing Properties from Topological Axioms
www.igi-global.com/chapter/exceptions-ontologies-theoretical-model-deducing/62435?camid=4v1a

Selection of Representative Feature Training Sets With Self-Organized Maps for Optimized Time Series Modeling and Prediction: Application to Forecasting Daily Drought Conditions With ARIMA and Neural Network Models

Synthesis of Flexible Fault-Tolerant Schedules for Embedded Systems with Soft and Hard Timing Constraints
www.igi-global.com/chapter/synthesis-flexible-fault-tolerant-schedules/51395?camid=4v1a