Chapter 8
Model-Driven Reengineering

Ricardo Pérez-Castillo
University of Castilla-La Mancha, Spain

Ignacio García Rodríguez de Guzmán
University of Castilla-La Mancha, Spain

Mario Piattini
University of Castilla-La Mancha, Spain

ABSTRACT

Legacy information systems entail a risk for companies because, on the one hand, they cannot be thrown away since valuable business knowledge becomes embedded in them over time, and on the other hand, they cannot be easily maintained at a moderate cost. Over the last two decades, reengineering has been the solution to these problems, since it supports the evolutionary maintenance of legacy information systems whilst simultaneously preserving the knowledge embedded within them. Unfortunately, traditional reengineering is facing new challenges concerning its formalization and automation as a consequence of legacy information systems being increasingly larger and more complex. A new software engineering approach known as Model-Driven Reengineering has consequently emerged to deal with these limitations. Model-Driven Reengineering does not replace traditional reengineering, but incorporates the model-driven development principles; i.e., this approach treats all software artifacts as models and establishes transformations between these models at different degrees of abstraction. The objective of this chapter is to provide an overview of the emerging concepts and standards related to Model-Driven Reengineering. This chapter also discusses how Model-Driven Reengineering deals with typical challenges that emerge when LISs are evolved, in order to mitigate the negative effects of the software erosion phenomenon, preserve the embedded business knowledge, and reduce maintenance costs.

DOI: 10.4018/978-1-61350-438-3.ch008

INTRODUCTION

Although software is an intangible object, the quality of software diminishes over time in a similar way to that of material objects. Lehman's first law states that an information system must continually evolve or it will become progressively less suitable in a real-world environment (Lehman et al., 1998). Companies currently have an enormous amount of large legacy systems which undergo the phenomenon of software erosion and software ageing. This means that existing information systems become progressively less maintainable (Polo et al., 2003). The negative effects of software erosion can be dead code, clone programs, missing capacities, inconsistent data and control data (coupling), among others (Visaggio, 2001).

On the one hand, software maintenance is part of the software erosion problem, since software erosion is due to maintenance itself and to the uncontrolled evolution of the system over time. On the other hand, software maintenance is also part of the solution to software erosion. The successive changes in information systems transform them into Legacy Information Systems (LIS), and a new and improved system must therefore replace the previous one when the maintainability levels diminish below acceptable limits (Mens, 2008). Nevertheless, the wide replacement of these systems from scratch is a key challenge since it makes a great impact on the technological, human and economic aspects of companies (Sneed, 2005). Firstly, the entire replacement of LISs affects technological and human aspects, since it usually involves retraining all the users in order for them to understand the new system and/or the new technology. Secondly, the new system may have a lack of specific functionalities that are missing as a result of technological changes. Thirdly, the economic aspect of companies is also affected, since the replacement of an entire LIS, by implementing a new system from scratch, implies a low Return of Investment (ROI) with regard to the old system. In addition, the development or purchase of the new system might exceed a company's budget.

In order to understand why a complete replacement from scratch in not an appropriate solution to the software erosion phenomenon, the following example, adapted from (Pérez-Castillo et al., 2011), is provided: Let us imagine a transmission belt in a car engine. This piece deteriorates progressively over time. When this piece is damaged, or its quality decreases considerably, it may become a threat to the overall performance of the motor. This transmission belt must consequently be replaced immediately, and the engine will therefore operate normally after the replacement. The solution in this case is easy, but an information system used in a company is more complicated. When an information system ages, it cannot simply be replaced by another new system for two important reasons: (i) a transmission belt costs a few dollars while an enterprise information system costs thousands of dollars, but in addition, (ii) while the environment of the belt (i.e. the motor) does not change, a considerable amount of business knowledge becomes embedded in the aged system over time in order to address the changes in the company's environment. This embedded knowledge is lost if the aged information system is replaced in its entirety, since this knowledge is not present anywhere else. A company with a new system may not therefore work normally, unlike the car engine.

An alternative to an entire replacement from scratch is another solution to software erosion that provides better results: software evolution. Software evolution is a kind of software maintenance which is also termed as evolutionary maintenance. In general, the maintenance process can perform four categories of modifications in the existing software (ISO/IEC, 2006):

- **Corrective maintenance**, which modifies a software product after delivery in order to correct any problems discovered.