Chapter 4
Architecture-Driven Modernization

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 can be a serious headache for companies because, on the one hand, these systems cannot be thrown away since they store a lot of valuable business knowledge over time, and on the other hand, they cannot be maintained easily at an acceptable cost. For many years, reengineering has been a solution to this problem because it facilitates the reuse of the software artifacts and knowledge embedded in the system. However, reengineering often fails due to the fact that it carries out non-standardized and ad hoc processes. Currently, software modernization, and particularly ADM (Architecture-Driven Modernization), standardized by the OMG, is proving to be an important solution to that problem, since ADM advocates carrying out reengineering processes taking into account the principles and standards of model-driven development. This chapter provides an overview of ADM and shows how it allows legacy information systems to evolve, making them more agile, preserving the embedded business knowledge, and reducing maintenance costs. Also, this chapter presents the software archeology process using ADM and some ADM success stories.

INTRODUCTION

According to the Lehman's first law, an information system must continually evolve or it will become progressively less suitable in real-world environments (Lehman et al., 1998). Indeed, companies count on a vast number of large legacy systems which are not immune to software erosion and software ageing, i.e., legacy information systems that become progressively less maintainable (Polo et al., 2003). Nevertheless, software erosion is due to maintenance itself and the evolution of

DOI: 10.4018/978-1-60960-215-4.ch004

the system over time. It is possible to measure this erosion using different metrics (Visaggio, 2001), e.g., dead code, clone programs, missing capacities, inconsistent data and control data (coupling), among others.

The successive changes in an information system degrade its quality, and thus, a new and improved system must replace the previous one. However, the wholesale replacement of these systems from scratch is risky since it has a great impact in technological, human and economic terms (Koskinen et al., 2004; Sneed, 2005). The technological and human point of view is affected since replacement would involve retraining all the users in order to understand the new system and the new technology, or the new system may lack specific functionalities that are missing due to the technological changes. Moreover, the economic point of view is also affected since the replacement of an entire legacy system implies a low Return of Investment (ROI) in that system. In addition, the development or purchase of a new system could exceed a company's budget.

For example, let us imagine a transmission belt in a car engine which deteriorates over use and over time. When this piece is damaged, it must be replaced, and then the engine operates normally. This example is easy, but an information system used in a company is more difficult. When this system ages, it cannot simply be replaced by another new system for two important reasons: (i) a belt costs a few dollars while an enterprise information system costs thousands of dollars, but in addition (ii) the aged system embeds a lot of business knowledge over time that is lost if it is replaced, thus the company with a new system may not operate normally like the car engine.

When companies are faced with the phenomenon of software erosion, evolutionary maintenance is a better solution to obtain improved systems, without discarding the existing systems, thus minimizing the software erosion effects. Evolutionary maintenance makes it possible to manage controllable costs and preserves the valuable business knowledge embedded in the legacy system, since 78% of maintenance changes are corrective or behaviour-preserving (Ghazarian, 2009).

Over the last two decades, reengineering has been the main tool for addressing the evolutionary maintenance of legacy systems (Bianchi et al., 2003). Reengineering preserves the legacy knowledge of the systems and makes it possible to change software easily, reliably and quickly, resulting in a maintenance cost that is also tolerable (Bennett et al., 2000).

The reengineering is the examination and alteration of a subject system to reconstitute it in a new form and the subsequent implementation of the new form […] This may include modifications with respect to new requirements not met by the original system.(Chikofsky et al., 1990)

Nevertheless, a 2005 study states that over 50% of reengineering projects fail (Sneed, 2005). This is due to the fact that in most cases the reengineering usually has two main problems when dealing with specific challenges at this point in time:

- the reengineering of large complex legacy information systems is very difficult to automate (Canfora et al., 2007), therefore the maintenance cost grows significantly.
- the traditional reengineering processes lacks formalization and standardization (Kazman et al., 1998), and thus different reengineering tools that address specific tasks in the reengineering process cannot be integrated or reused in different reengineering projects.

For these reasons, the software industry is demanding reengineering processes that enable the evolutionary maintenance of legacy systems in an automatic and standardized way. The typical reengineering concept has shifted to so-called Architecture-Driven Modernization (ADM) as a solution to those demands.