Model-Driven Software Modernization

S

Liliana Maria Favre
Universidad Nacional Del Centro De La Provincia De Buenos Aires, Argentina

Liliana Martinez
Universidad Nacional del Centro de la Provincia de Buenos Aires, Argentina

Claudia Teresa Pereira
Universidad Nacional del Centro de la Provincia de Buenos Aires, Argentina

INTRODUCTION

Most vital software systems in companies have been developed several years ago with technology that today can be considered obsolete and unaligned with their current strategic objectives. These legacy systems plays a central role in the company´s information system. They are often large business-critical applications that have involved the investment of money, time and other resources through the years. Therefore, companies are facing the problematic of having to modernize or replace their legacy software systems.

Unlike the maintenance, software modernization involves significant changes in the structure and functionality of a legacy system with the aim of increasing its strategic value. Software modernization, understood as technological and functional evolution of legacy systems, provides principles, methods, techniques and tools to support the transformation from an existing software system to a new one that satisfies new requirements. It is related to reverse engineering, restructuring, forward engineering and reengineering processes. The definitions given by Chikofsky and Cross (1990) for these processes are still in force beyond the technological evolution:

- *Reverse engineering* is the process of analyzing a subject system to identify the system's components and their interrelationships and, create representations of the system in another form or at a higher level of abstraction.
- *Restructuring* is the transformation from one representation form to another at the same relative abstraction level while preserving the subject system´s external behavior (functionality and semantic).
- *Forward engineering* is the traditional process of moving from high-level abstractions and logical, implementation-independent designs to the physical implementation of a system.
- *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.

The term *Refactoring*, introduced by Martin Fowler, defines a special kind of restructuring in object-oriented code (Fowler, 1999). One of the advantages of software modernization versus traditional software reengineering is that the former allows discovering and refactoring object-oriented models at a high level of abstraction, such as business process models.

Software modernization faces many challenges due to the proliferation of new technologies such as mobile computing and cloud computing. Thus, novel technical frameworks for information integration and tool interoperability are needed. The *Model Driven Development* (MDD) appears to be an interesting approach to address these chal-

DOI: 10.4018/978-1-5225-2255-3.ch648

lenges since it provides principles and techniques to represent software systems through models at different levels of abstraction. MDD refers to a range of development approaches based on the use of software models as first class entities. The most well-known realization of MDD is the OMG standard Model Driven Architecture (MDA) (MDA, 2016). The outstanding ideas behind MDA are separating the specification of the system functionality from its implementation on specific platforms, managing the software evolution from abstract models to implementations increasing the degree of automation of model transformations and achieving interoperability with multiple platforms, programming languages and formal languages. The essence of MDA is the Meta Object Facility Metamodel (MOF) that allows different kinds of software artifacts to be used together in a single project (MOF, 2015). Models play a major role in MDA which distinguishes Computation Independent Model (CIM), Platform Independent Model (PIM) and Platform Specific Model (PSM). Some authors also distinguish Implementation Specific Model (ISM) as a description (specification) of the system in source code. To express transformations, OMG defined the MOF 2.0 Query, View, Transformation (QVT) metamodel (QVT, 2015).

OMG Architecture-Driven Modernization Task Force (ADMTF) is developing a set of specifications and promoting industry consensus on modernization of existing applications (ADM, 2016). In the ADM context, software modernization can be summarized as follows. First, information is extracted out of the system artifacts. Second, this information is analyzed in order to take adequate modernization decisions and finally, the information must be transformed to new artifacts. These steps are supported by metamodels to describe existing systems, discoverers to automatically create models of these systems and, tools to understand and transform complex models created out of existing systems. The *Knowledge Discovery Metamodel (KDM*) and the *Abstract Syntax Tree Metamodel (ASTM*) are two complementary and relevant ADM standards (KDM, 2011) (ASTM, 2011).

This chapter analyzes ADM-based software modernization. It provides an overview of the-state-of-the-art in software modernization techniques. Taxonomy of different techniques is described. A description of how traditional techniques such as static and dynamic analysis can be integrated with ADM standards is presented. We propose a framework for ADM software modernization that integrates different paradigms. Finally, challenges and strategic directions in software modernization are included.

BACKGROUND

25 years ago, modernization focused mainly on reverse engineering for recovering high-level architectures or diagrams from procedural code to face up with problems such as comprehending data structures or databases or the Y2K problem. At that time, many different kinds of static analysis techniques had been developed and several studies had been carried out to compare them.

Later years, new approaches were developed to identify objects into legacy code and translate this code into an object-oriented language. Object-oriented programs are essentially dynamic and present particular problems linked to polymorphism and late binding, abstract classes and dynamically typed languages. Then, the focus of object-oriented software analysis moved from static analysis to dynamic one, more precisely static analysis was complemented with dynamic one. Many works had contributed to reverse engineering object-oriented code. Muller, Jahnke, Smith, Storey, Tilley, and Wong (2000) present a roadmap for reverse engineering research for the first decade of the 2000s. Fanta and Rajlich (1998) describe the reengineering of a deteriorated object-oriented industrial program written in C++. Systa (2000) describes an experimental environment to reverse engineer JAVA software