Chapter III

A Unified Software Reengineering Approach towards Model Driven Architecture Environment

Bing Qiao, De Montfort University, UK
Hongji Yang, De Montfort University, UK
Alan O’Callaghan, De Montfort University, UK

Abstract

When developing a software system, there are a number of principles, paradigms, and tools available to choose from. For a specific platform or programming language, a standard way can usually be found to archive the ultimate system; for example, a combination of an incremental development process, object-oriented analysis and design, and a well supported CASE (Computer-Aided Software Engineering) tool. Regardless of the technology to be adopted, the final outcome of the software development is always a working software system. However, when it comes to software reengineering, there is rather less consensus on either approaches or outcomes. Shall we use black-
box or white-box reverse engineering for program understanding? Shall we produce data and control flow graphs, or some kind of formal specifications as the output of analysis? Each of these techniques has its pros and cons of tackling various software reengineering problems, and none of them on its own suffices to a whole reengineering project. A proper integration of various techniques capable of solving a specific issue could be an effective way to unravel a complicated software system. This kind of integration has to be done from an architectural point of view. One of the most exciting outcomes of recent efforts on software architecture is the Object Management Group’s (OMG) Model-Driven Architecture (MDA). MDA provides a unified framework for developing middleware-based modern distributed systems, and also a definite goal for software reengineering. This chapter presents a unified software reengineering methodology based on Model-Driven Architecture, which consists of a framework, a process, and related techniques.

**Introduction**

What are the requirements of software reengineering nowadays? The requirements for software reengineering arise when the increasing business value outpaced the deteriorating maintainability of its underlying software infrastructure. The complexity of software maintenance has been aggravated by the introduction of the Internet giving rise to a great many popular, similar, but ultimately incompatible techniques (the side product of flexibility and openness) and gigantic software systems on the Web. It is often either too expensive or technically impossible to replace working systems on which new technologies can no longer be applied. Software maintenance on such systems becomes effectively limited to bug fixes or small functional enhancements that do not effect major structural changes. It is also difficult and expensive to find human expertise in the older technologies required to maintain legacy systems, and the accumulation of small changes inevitably results, over time, in a big impact. For such software systems, software reengineering is the only way to extend their operational lifetime or even make them capable of accommodating changes in a brand new form.

The first challenge of software reengineering is to understand a legacy system by producing system views at different abstraction levels, a task often hampered by the lack of documentation, and the immense and growing amount of legacy code. Reverse engineering is employed to identify the components of a system and their interrelationships, creating representations of the system in another form that is often at a higher level of abstraction. Program understanding can be achieved via reverse engineering, too. The two forms of reverse engineering techniques are described as: black box and white box. The former emphasises the external interfaces of subsystems, whilst the latter stresses a deep understanding of individual modules (Weiderman, 1997). The two forms of reverse engineering could be used together or separately. If the main purpose is to integrate a large legacy application into a new system, it is neither feasible nor necessary to understand the legacy application deeply. The black box reverse engineering should suffice. If only the business logic needs to be uncovered from lines of code, the white box could be applied for program understanding. However, in order to extract business
Related Content

Distributed Business Rules within Service-Centric Systems
[www.igi-global.com/chapter/distributed-business-rules-within-service/35870?camid=4v1a](www.igi-global.com/chapter/distributed-business-rules-within-service/35870?camid=4v1a)

Design Recovery of Web Applications Transactions
[www.igi-global.com/chapter/design-recovery-web-applications-transactions/4928?camid=4v1a](www.igi-global.com/chapter/design-recovery-web-applications-transactions/4928?camid=4v1a)

XML Benchmarking: The State of the Art and Possible Enhancements
[www.igi-global.com/chapter/xml-benchmarking-state-art-possible/27787?camid=4v1a](www.igi-global.com/chapter/xml-benchmarking-state-art-possible/27787?camid=4v1a)

Specification of Business Components Using Temporal OCL
[www.igi-global.com/chapter/specification-business-components-using-temporal/30537?camid=4v1a](www.igi-global.com/chapter/specification-business-components-using-temporal/30537?camid=4v1a)