Chapter VIII

Architectural Slicing to Support System Evolution

Jianjun Zhao, Fukuoka Institute of Technology, Japan
Limin Xiang, Kyushu Sangyo University, Japan

ABSTRACT

Change impact analysis is a useful technique in software maintenance and evolution. Many techniques have been proposed to support change impact analysis at the code level of software systems, but little effort has been made for change impact analysis at the architectural level. In this chapter, we present an approach to supporting change impact analysis at the architectural level of software systems based on the architectural slicing and chopping technique. The main feature of our approach is to assess the effect of changes in a software architecture by analyzing its formal architectural specification, and therefore, the process of change impact analysis can be automated completely.

INTRODUCTION

Software change is an essential operation for software evolution. The change is a process that introduces new requirements into an existing system, modifies the system if the requirements were not correctly implemented, or moves the system into a new operational environment. The mini-cycle of change during software evolution (Yau et al., 1978) is composed of several phases: request for change; planning phase, consisting of program comprehension and change impact analysis; change implementation, including restructuring for change and change propagation; verification and validation; and re-documentation.

Change impact analysis (Bohner & Arnold, 1996; Lindvall & Sandahl, 1998), one of the system analysis methods, is the task through which the programmers can assess the extent of the change, that is, the software component that will impact the change, or be
impacted by the change. Change impact analysis provides techniques to address the problem by identifying the likely ripple effect of software changes and using this information to re-engineer the software system design.

Change impact analysis may have different dimensions (regarding the levels of abstraction of a system) according to which the separation of concerns (Parnas, 1972) can be made. For example, change impact analysis can be performed at the code level to obtain the detailed information regarding the effect of changes at the code level of the system. And also, change impact analysis can be performed at the architectural level of a system to reduce the complexity of code-level change impact analysis and to allow a programmer of software maintenance to assess the effect of changes of the system structure at the architectural level so that software evolution actions can be made earlier (Rombach, 1990). The separation of change impact analysis into different levels is therefore a necessary step to reduce the cost of change impact analysis during software evolution.

In order to develop change impact analysis techniques at the architectural level of software systems to support architectural evolution during software design, formal modeling of software architectures is required. The software architecture of a system defines its high-level structure, exposing its gross organization as a collection of interacting components. A well-defined architecture allows an engineer to reason about system properties at a high level of abstraction (Shaw & Garlan, 1996). Architecture description languages (ADLs) are formal languages that can be used to represent the architecture of a software system. They focus on the high-level structure of the overall application rather than the implementation details of any specific source module. In order to support formal representation and reasoning of software architecture, a number of ADLs such as WRIGHT (Allen, 1997), Rapide (Luckham et al., 1995), and UniCon (Shaw et al., 1995) have been proposed. By using an ADL, a systems architect can formally represent various general attributes of a software system’s architecture. This provides a promising solution to developing techniques to support change impact analysis at the architectural level because formal language support for software architecture provides a useful platform on which automated support tools for architectural level impact analysis can be developed.

In this chapter, we focus our attention on the issue of the planning phase of the mini-cycle for software change during software evolution, in particular, the change impact analysis, to support architectural level evolution of software systems. We introduce an approach to supporting change impact analysis at the architectural level of software systems based on architectural slicing techniques (Zhao, 1997, 1998). The main feature of our approach is to assess the effect of changes in a software architecture by analyzing its formal architectural specification, and therefore, the process of change impact analysis can be automated completely.

Traditional program slicing, originally introduced by Weiser (1979), is a decomposition technique that extracts program elements related to a particular computation. A program slice consists of those parts of a program that may directly or indirectly affect the values computed at some program point of interest, referred to as a slicing criterion. The task of computing program slices is called program slicing. As shown in Bohner and Arnold (1996), program slicing is an essential technique to support change impact analysis of software systems at the source code level. In contrast to traditional program slicing, architectural slicing is designed to operate on a formal architectural specification.
The Development of ICT for Envisioning Cloud Computing and Innovation in South Asia
www.igi-global.com/chapter/development-ict-envisioning-cloud-computing/74135?camid=4v1a