Chapter 9
An Integrated Process for Aspect Mining and Refactoring

Esteban S. Abait
UNICEN University, Argentina

Santiago A. Vidal
UNICEN University, Argentina

Claudia A. Marcos
UNICEN University, Argentina

Sandra I. Casas
UARG - Universidad Nacional de la Patagonia Austral, Argentina

Albert A. Osiris Sofia
UARG - Universidad Nacional de la Patagonia Austral, Argentina

ABSTRACT

Aspect-Oriented Software Development (AOSD) aims at solving the problem of encapsulating crosscutting concerns, which orthogonally crosscut the components of a system, in units called aspects. This encapsulation improves the modularization of a system and in consequence its maintenance and evolution. In this work, the authors propose a systematic process for the migration of object-oriented systems to aspect-oriented ones. This migration is achieved in two main phases: crosscutting concern identification (aspect mining) and code transformation (aspect refactoring). The aspect mining phase is based on dynamic analysis and association rules to identify potential crosscutting concerns. The aspect refactoring phase, on the other hand, uses inference rules to identify the refactoring that can be applied. The whole process is described and its application on a real system is assessed.

INTRODUCTION

One of the main problems that software developers have to deal with is the unexpected and continuous evolution inherent to all software systems. This evolution implies that successful software systems must change or become less satisfactory (Lehman & Belady, 1974). Improving the separation of concerns (Parnas, 1972) in those software systems is likely to ameliorate their adaptability for changing
environments. This separation of concerns is of vital importance in order to avoid many of the problems that complicate software engineering: degradation of software comprehensibility, invasive modifications due to maintenance and evolution tasks, reduced reusability of the developed artifacts and limited traceability (Tarr, Ossher, Harrison, & Sutton, 1999).

Aspect-Oriented Programming (AOP) (Kiczales et al., 1997) is a programming paradigm that promotes the separation of concerns in software systems. Particularly, AOP provides mechanism for the correct encapsulation of crosscutting concerns. Such concerns crosscut the dominant decomposition of the application restricting the separation of concerns on software. For instance, tracing, persistence or synchronization are common examples of crosscutting concerns. Code related to these crosscutting concerns can show two symptoms of bad modularization: it can be scattered over the whole project or it can be tangled with other code (Hannemann & Kiczales, 2001). AOP introduces a new modularization unit, called aspect, whose goal is the encapsulation of these crosscutting concerns.

The migration of an object-oriented software system into an aspect-oriented one is far from being a trivial task. Due to the large size of the implementation and the lack of sound documentation there is a need for tools that automate the identification, quantification and refactorization of crosscutting concern into aspects of the new systems. A first step towards achieving this goal is to discover the different crosscutting concerns present in the code, in order to decide whether they are aspect candidates for a future system. The task of identifying the crosscutting concerns that are amenable for an aspect-oriented implementation is called aspect mining (Kellens, Mens & Tonella, 2007). Transforming those crosscutting concerns to aspects of the new systems is regarded as aspect refactoring (Kellens, Mens & Tonella, 2007).

This work presents a comprehensive approach to perform the gradual evolution of an object-oriented system to an aspect-oriented one. The proposed approach aims at assisting the developer in: performing the evolution process, automating most tasks involved in this process, taking advantage of precise aspect mining techniques, and applying different types of aspect refactorings. We argue that the migration from an OO system to an AO one improves the structure and quality of the software, and thus eases software evolution. Besides, we believe that the provision of semi-automated support to help the developer to discover crosscutting concerns and to encapsulate them into aspects is beneficial. A novelty of the proposed approach is the use of dynamic analysis together with data mining techniques for identifying candidate aspects. In addition, an aspect refactoring process based on existing types of refactorings is presented, which automates the major steps of the migration.

The chapter is organized as follows. Firstly, the whole process is presented and its different steps are detailed. Next, the aspect mining and refactoring techniques are introduced and explained through a toy example. The following section presents the results of applying the full migration process on real software system. Afterwards, the related work is presented and the conclusion and future works are drawn.

THE APPROACH

The proposed approach consists of two main phases (Figure 1): (i) aspect mining, and (ii) aspect refactoring. The first phase receives an object-oriented system (to be evolved) as input, and produces a number of candidate aspects as output. These aspects are identified by making a dynamic analysis of the system and applying association rules. The information of candidate aspects and the initial system’s source code are then passed to the aspect refactoring phase. In this phase, different refactorings are evaluated and eventually applied to the code. As output,