Software Evolution and Aspect-Oriented Programming

Tom Mens, Kim Mens, Tom Tourwé

October 15, 2004

Abstract

All software systems are subject to evolution. This poses great challenges to software engineers. We claim that aspect-oriented software development (AOSD), in addition to separating the different concerns during software development, can be seen as a way to overcome many of the problems related to software evolution. At the same time, AOSD may benefit from tools and techniques that automate software evolution. Our research explores the cross-fertilisation between these two active research domains.

Introduction

Real world Software needs to evolve continually in order to cope with ever-changing software requirements. This characteristic has been identified by Manny Lehman in his so-called first law of software evolution, addressing **continuing change**: *A program that is used must be continually adapted else it becomes progressively less satisfactory.*

This need for software to evolve continuously poses important challenges on software engineers. Advanced automated software engineering techniques and tools are needed to improve software evolution support. An *ERCIM Working group on Software Evolution* was launched in May 2004 to address this need. Two important techniques that will be investigated actively are *software restructuring* and *aspect-oriented software development*.

Software Restructuring

Software restructuring should be an essential activity in software engineering, according to Lehman's second law of software evolution, addressing **increasing complexity**: *As a program is evolved its complexity increases unless work is done to maintain or reduce it.*

In program transformation research, two different restructuring approaches can be distinguished. The term *rephrasing* is used to refer to techniques that improve the structure of the software without changing the implementation language. A typical example is *software refactoring*, which aims at improving the internal structure of a program without changing its external behaviour. The term *translation* refers to techniques that restructure the software across programming languages. A typical example is *migration* of legacy code to an object-oriented equivalent (e.g., COBOL to Java).

Aspect-Oriented Software Development

An essential problem with software development is the *tyranny of the dominant decomposition*. No matter how well a software system is decomposed into modular units, there will always be *concerns*

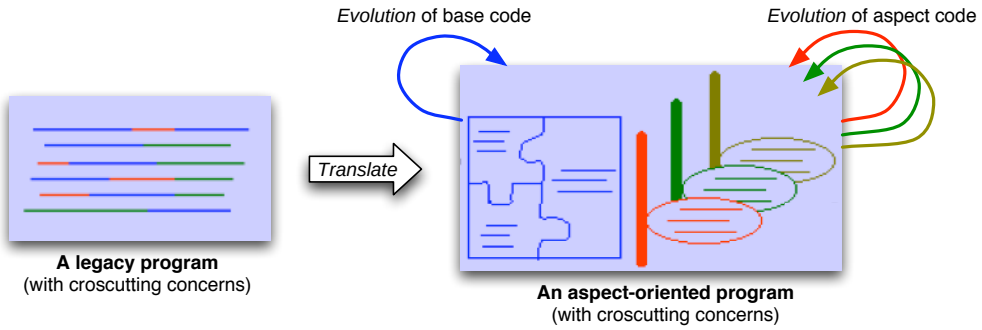

Figure 1: Cross-fertilisation of software evolution and AOSD

(typically non-functional ones) that cut across the chosen decomposition. The code of these cross-cutting concerns will necessarily be spread over different modules, which has a negative impact on the software quality in terms of comprehensibility, adaptability and evolvability.

Aspect-oriented software development (AOSD) has been proposed as a solution to this problem. In order to capture such cross-cutting concerns in a localised way, a new abstraction mechanism (called an *aspect*) is added to existing programming languages (e.g., AspectJ for Java). As a result, cross-cutting concerns are captured in a localised way and are no longer distributed over different modules. This makes the software more maintainable, evolvable and understandable.

Cross-fertilisation

In order for AOSD to become truly succesful, we need to translate existing software systems into their aspect-oriented equivalents and rephrase them continuously (Figure 1). Given the size and complexity of industrial software systems, this should be achieved with as much automated support as possible. More specifically, we need automated support for three essential activities.

Aspect mining techniques should be used to identify the relevant concerns in the source code.

Aspect introduction techniques are needed to define the appropriate aspects for any of the identified concerns, in order to restructure the base code in an aspect-oriented way.

Aspect evolution techniques are required in order to evolve an aspect-oriented application.

Our research investigates how formal techniques, successful in supporting traditional software evolution, can support these three new activities. For example, in a recent experiment, we used *formal concept analysis* to mine for *crosscutting concerns* in the source code. With this approach we detected interesting features corresponding to crosscutting functional or non-functional concerns, as well as occurrences of design patterns like the *Visitor* design pattern (see Figure 2). Typically, the implementation of such a pattern spans multiple classes and methods, and grouping these in a single view allows the developer to understand and manipulate the pattern more easily.

Another initial experiment deals with aspect introduction and evolution, in particular with how evolution of the base code affects the definition of aspects that work on it. In order to deal with this issue, we proposed a more sophisticated aspect-oriented programming language, accompanied by an advanced development environment. The environment helps a developer to define aspects, and is able

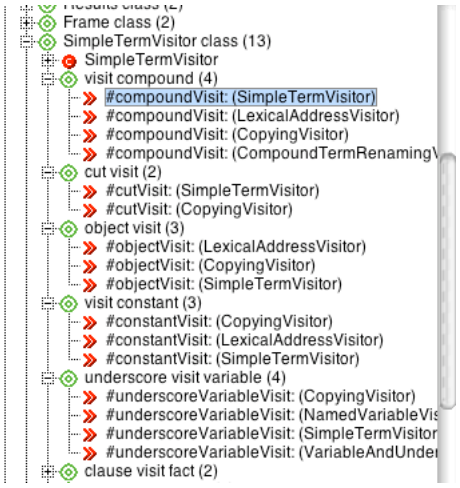

Figure 2: An occurrence of the Visitor Design Pattern detected by Formal Concept Analysis

to assesses the impact of evolution on these aspects automatically. To this extent, the environment uses a machine learning technique called inductive logic programming.

European collaboration

The authors are involved in many successful international research activities that have been initiated in the domains discussed above:

- A Belgian FWO-funded scientific research network on “Foundations of Software Evolution” (prog.vub.ac.be/FFSE/network.html)
- A Belgian IWT-funded interuniversity research project on “Architectural Resources for the Restructuring and Integration of Business Applications” (arriba.vub.ac.be)
- A Belgian FNRS-funded “Research Center on Structural Software Improvement”
- A European ESF-funded scientific network on “Research Links to Explore and Advance Software Evolution” (labmol.di.fc.ul.pt/projects/release/)
- A European EU-funded network of excellence on “Aspect-Oriented Software Development”
- An ERCIM working group on “Software Evolution”

Please contact

Kim Mens, Dpartement d’Ingnierie Informatique, Universit catholique de Louvain

E-mail: Kim.Mens@info.ucl.ac.be