Refactoring Object-Oriented Systems with Aspect-Oriented Concepts

Extended Abstract for the **II Simpósio Doutoral do DI**

December 2004

Ph.D. student:
Miguel Pessoa Monteiro
mmonteiro@di.uminho.pt

Ph.D. supervisor:
João Miguel Fernandes
jmf@di.uminho.pt

Departamento de Informática, Escola de Engenharia, Universidade do Minho
Campus de Gualtar, 4710-057 Braga, Portugal

Software engineering tools and languages should support complete separation of concerns, by enabling the deployment of source code relative to each different concern in its own unit of modularity. Unfortunately, current tools and languages – including those supporting the object-oriented programming (OOP) paradigm – fail to provide a complete and effective support for full separation of all concerns. Undesirable phenomena such as code scattering and code tangling ensue.

Advanced Separation of Concerns (ASoC) is a new area of research whose purpose is to develop tools and languages that enable an effective separation of all concerns. Aspect-oriented programming (AOP) is one such field, which complements existing programming paradigms, including OOP, with constructs that effectively provide full separation of concerns. AOP and its main language, AspectJ, are receiving an ever-wider acceptance and adoption.

Refactoring is a technique that aims to restructure the source code of applications in order to improve its underlying design while preserving its functionality, or "externally observable behaviour". Refactoring reverses the more traditional order of design first, code next, and its purpose is to enable applications to evolve in time, according to changing environments and requirements. Inadequate structures and designs are detected through "code smells" – symptoms in the source code that may be indicative of something that may be unsuitable, inadequate, or plain wrong in the source code. Code smells are gradually removed from code bases through refactoring processes, comprising sequences of small, behaviour-preserving transformations in the source code. The name "refactoring" is also used to refer to these code transformations. Larger refactorings are based on sequences of smaller refactorings. Adequate coverage of unit tests is a fundamental prerequisite for any refactoring process.

The prospect of AOP becoming a mainstream technology in the near future begs the question of how to deal with a large base of OOP legacy code. AspectJ's backward compatibility with Java opens the way for refactoring existing Java applications in order to leverage the concepts and mechanisms of aspects. This requires first a clear idea of what comprises a "good style" for AOP languages and in particularly what distinguishes it from good OOP style. Experience in the last half-decade suggests that pattern languages of refactorings and code smells are an effective way to represent and characterise notions of good style.

In this Ph.D. project we propose to find novel refactorings and code smells for representing a new style, suitable for AOP code. Such refactorings and code smells are to be used on AOP code and for restructuring existing OOP legacy source code so they can benefit from aspects. We took the approach of using suitable case studies to provide useful insights.

As it nears completion, the research carried out in the context of this Ph.D. project yielded (1) a collection of 28 different refactorings for AOP source code, (2) a review of traditional code smells in light of aspect-orientation, and (3) several novel code smells. Most of these smells focus on symptoms in OOP source code that are possible indicators of latent aspects. One of the proposed code smells is specific to aspects. We also validate the refactorings with an illustrative refactoring process using 17 of the refactorings, which targets a Java code base.

Keywords: Aspect-Oriented Programming, Refactoring, Object-Oriented Programming.