Chapter VIII

Model-Driven Software Refactoring

Tom Mens
University of Mons-Hainaut, Belgium

Gabriele Taentzer
Philipps-Universität Marburg, Germany

Dirk Müller
Chemnitz University of Technology, Germany

ABSTRACT

In this chapter, we explore the emerging research domain of model-driven software refactoring. Program refactoring is a proven technique that aims at improving the quality of source code. Applying refactoring in a model-driven software engineering context raises many new challenges such as how to define, detect and improve model quality, how to preserve model behavior, and so on. Based on a concrete case study with a state-of-the-art model-driven software development tool, AndroMDA, we explore some of these challenges in more detail. We propose to resolve some of the encountered problems by relying on well-understood techniques of meta-modeling, model transformation and graph transformation.

INTRODUCTION

In the current research and practice on software engineering, there are two very important lines of research for which tool support is becoming widely available. The first line of research is program refactoring, the second one is model-driven software engineering. To this date, however, the links and potential synergies between these two lines of research have not been sufficiently explored. This will be the main contribution of this chapter.

Model-Driven Software Engineering

In the realm of software engineering, we are witnessing an increasing momentum towards the use of models for developing software systems. This trend commonly referred to as model-driven software engineering, emphasizes on models as the primary artifacts in all phases of software
Model-Driven Software Refactoring

development, from requirements analysis over system design to implementation, deployment, verification and validation. This uniform use of models promises to cope with the intrinsic complexity of software-intensive systems by raising the level of abstraction, and by hiding the accidental complexity of the underlying technology as much as possible (Brooks, 1995). The use of models thus opens up new possibilities for creating, analyzing, manipulating and formally reasoning about systems at a high level of abstraction.

To reap all the benefits of model-driven engineering, it is essential to install a sophisticated mechanism of model transformation, that enables a wide range of different automated activities such as translation of models (expressed in different modeling languages), generating code from models, model refinement, model synthesis or model extraction, model restructuring etc. To achieve this, languages, formalisms, techniques and tools that support model transformation are needed. More importantly, their impact on the quality and semantics of models needs to be better understood.

Program Refactoring

Refactoring is a well-known technique to improve the quality of software. Martin Fowler (1999) defines it as “A change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior”.

The research topic of refactoring has been studied extensively at the level of programs (i.e., source code). As a result, all major integrated software development environments provide some kind of automated support for program refactoring.

As a simple example of a program refactoring, consider the refactoring Extract Method, one of the more than 60 refactorings proposed by Fowler. Essentially, it is applied to a method in which part of the method body needs to be extracted into a new method that will be called by the original one. The situation before this program refactoring on a piece of Java source code is shown in Figure 1, the situation after is shown in Figure 2. The code lines that differ between both versions are marked with an asterisk.

For program refactoring, a wide variety of formalisms has been proposed to gain a deeper understanding, and to allow formal analysis. One of these formalisms is graph transformation theory (Mens et al., 2005). We mention it here explicitly, as we will show later in this chapter how this formalism can be applied to support model refactoring as well. It is, however, not our goal to provide a detailed overview of existing work on program refactoring here. For the interested reader, we refer to a detailed survey of the state-of-the-art in this domain (Mens & Tourwé, 2004).

Model-Driven Software Refactoring

A natural next step seems to explore how the idea of refactoring may be applied in a model-driven software development context. We will refer to this combination as model-driven software refactoring and we will explore the ramifications of this synergy in the current chapter.

One of the straightforward ways to address refactoring in a model-driven context is by raising refactorings to the level of models, thereby introducing the notion of model refactoring, which is a specific kind of model transformation that allows us to improve the structure of the model while preserving its quality characteristics. To the best of our knowledge, Sunyé et al. (2001) were the first to apply the idea of refactoring to models expressed in the Unified Modeling Language (UML).

As a simple yet illustrative example of a UML model refactoring is shown in Figure 3. It depicts a class model in which two classes having attributes of the same type have been identified. The model refactoring consists of removing the redundancy
Related Content

Unsupervised Estimation of Facial Expression Intensity for Emotional Scene Retrieval in Lifelog Videos

An Experimental Comparison of System Diagrams and Textual Use Cases for the Identification of Safety Hazards

Designing Resource-Constrained Embedded Heterogeneous Systems to Cope with Variability
[www.igi-global.com/chapter/designing-resource-constrained-embedded-heterogeneous-systems-to-cope-with-variability/116105?camid=4v1a](www.igi-global.com/chapter/designing-resource-constrained-embedded-heterogeneous-systems-to-cope-with-variability/116105?camid=4v1a)

Automatic Recognition of Traffic Signs with 3D Distance Estimation for Intelligent Vehicles