Correct Refactorings

Volker Stolz and Erlend Kristiansen

Institutt for Informatikk,
Universitetet i Oslo, Norge

{stolz,erlenkr}@ifi.uio.no

Abstract

According to Fowler[1], the term "refactoring" refers to "a change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior."

Especially in the setting of object-oriented programs, we can distinguish changes to the *structure* of the classes and methods (e.g. moving methods up/down the inheritance tree), and changes *within* a method (e.g. extracting a sequence of statements into a new method).

Software development tools and integrated development environments like Eclipse implement various (so-called) refactorings, but often do not enforce that the behaviour of the code does not change, even in obvious cases—and we might say, for obvious reasons.

We present an example, and motivate a mitigation against changed behaviour after a refactoring by introducing assertions during the refactoring that capture the intended semantics before the refactoring. We discuss the phenomen in a larger software engineering context, and give an outline how to validate our idea in practice.

[1] Martin Fowler, Kent Beck, John Brant, William Opdyke, Don Roberts. Refactoring: Improving the Design of Existing Code. Addison - Wesley, 1999.