Safer Refactorings

November 2014

candidate	Joakim Kristiansen
supervisors	Martin Steffen, Volker Stolz (UiO/HiB)
group	PMA
type	60 ECTS
recommended background	program analysis, refactoring, compilation
study program	computer science

Short description

The task is to design and implement semantic-preserving refactorings.

Background and motivation

Refactorings [1] are a common tool of the trade for programming. In general, they are assumed to be equivalent transformations of a program, to improve the quality of the source code with regard to a concrete or subjective software metric, such as coupling.

Problem setting

Alas, some seemingly intuitive refactorings do *not* preserve the original semantics. Especially for OO programs, proving a particular refactoring as correct is a complex task, e.g. in the presence of virtual method calls. We look at particular refactorings, and suggest to encode the assumptions necessary for their correctness in the form of assertions. While this does not directly preclude the developer from applying "wrong" refactorings, it makes the necessary assumptions explicit. A simple example:

```
class C                                  class C        class X
  X x;              transform into       X x;
   main() {        ==============>       main() {         f() {
     x.m();                                 x.f();           m(); n();
     x.n(); }                             }                }
```

To ascertain that $m()$ does not change the attribute 'x' of the calling object c (which would mean that the original $x.n()$ goes to a different target from $x.m()$, we introduce an additional parameter that is used in the newly introduced method f to assert the intended behaviour:

```
main() { x.f(this) }
f(C that) { m(); assert that.x == this; n() }
```

This transformation does not prevent erroneous applications of the transformation, but allows to reason about it (via 3rd party systems like the JML tools, or the KeY system), or develop a sufficient number of test cases. This improved refactoring could e.g. be integrated Eclipse's LTK Refactoring for Java programs, and contributed to the Open Source community. Future work could include source code repository mining (e.g. CVS/SVN/... repositories of open source software projects), and finding out if a particular refactoring has been applied, and give an indication as to its correctness. Some of the necessary infrastructure (code for analysis of Java fragments within Eclipse, finding candidates through heuristics etc.) has been used in the Master's thesis [3].

References

[1] M. Fowler. *Refactoring: Improving the Design of Existing Code*. Object Technology Series. Addison-Wesley, June 1999.

[2] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns*. Addison-Wesley, 1994.

[3] E. Kristiansen. Automated composition of refactorings. implementing and evaluating a search-based extract and move method refactoring. Master's thesis, University of Oslo, Dept. of Informatics, 2014.

[4] M. Schäfer, T. Ekman, and O. [de Moor. Challenge proposal: Verification of refactorings. In *Proceedings PLPV"09*, pages 67–72, New York, NY, USA, 2008. ACM.