A closer look at software refactoring using symbolic execution*

Csaba Szabó[a], Maroš Kotuľa[b], Richard Petruš[c]

Department of Computers and Informatics
Faculty of Electrical Engineering and Informatics
Technical University of Košice
[a]Csaba.Szabo@tuke.sk, [b]Maros.Kotula@student.tuke.sk,

[c]Richard.Petrus@student.tuke.sk

Abstract

Fowler's classical definition of program refactoring as it appeared in [1] is many times broken in CASE tools, which claim non-refactorings to be refactoring operations. On other hand-side, there are often forgotten refactorings. According to the original definition [1], "Refactoring is the process of changing a software in a such way that it does not alter the external behavior of the code yet improves its internal structure." Many times, refactoring is done by hand, which needs a detailed verification. Based on the commutativity diagram of symbolic execution presented by King in [2] and the above definition of refactoring, we expressed the principle of program refactoring and our look at it using symbolic execution. These are presented in this paper as well as the overall principle of automated refactoring evaluation. For the question of "is it a refactoring?" we offer an answer using Java PathFinder and Symbolic PathFinder for the Java programming language.

Keywords: Java, refactoring, symbolic execution

MSC: 68N15; 68N20

References

[1] Fowler, M., *Refactoring – Improving the Design of Existing Code*, Addison-Wesley Professional, 1st ed. (1999).

[2] King, J. C., Symbolic Execution and Program Testing, *Communications of the ACM* Vol. 19 No. 7 (1976).

*This work was supported by the Cultural and Educational Grant Agency of the Slovak Republic, Project No. 050TUKE-4/2013.