# Reduction of Regression Tests for Erlang Based on Impact Analysis

**István Bozó, Melinda Tóth, Zoltán Horváth**

Eötvös Loránd University

{bozoistvan,tothmelinda,hz}@elte.hu

**Abstract**

Legacy codes are changed in software maintenance processes to introduce new functionality, modify existing features, eliminate bugs etc. or by refactorings while the main original properties and the behaviour of the system should be preserved. Developers apply regression testing with highest degree of code coverage to be sure about it, and thus they retest the software after some modifications. Our research focuses on impact analysis of changes in applications written in the dynamically typed functional programming language, Erlang. To calculate the affected program parts, we use dependence graph based program slicing, therefore we have defined the Dependence Graphs with respect to the semantics of Erlang. Applying the results, we may shrink the set of test cases selected for regression testing for ones which are affected by the changes.

Impact analysis is a mechanism to find those source code parts that are affected by a change on the source code, therefore it could help in test case selection for regression testing.

Erlang [4] is a dynamically typed functional programming language that was designed for building concurrent, reliable, robust, fault tolerant, distributed systems with soft real-time characteristic like telecommunication applications. The language has become widespread in industrial applications in the last decade.

Our research focuses on selecting the test cases of the Erlang applications that are affected by a change on the source code. In other words, we want to calculate the impact of a source code modification. To calculate the affected program parts, we use dependence graph based program slicing [8, 6], therefore we have to define the Dependence Graphs for Erlang.

Refactoring [5] is the process of changing and improving the quality of the source code without altering its external behaviour. Refactoring can be done manually or using a refactoring tool. We have been developing a refactoring tool for Erlang, called RefactorErl [3]. RefactorErl is a source code analysis and transformation tool [2]. It provides 24 refactoring steps for Erlang developers, such as moving, renaming different language entities, altering the interface of functions or the structure of expressions, parallelisation, etc. Besides transformations, RefactorErl has different features to support code comprehension [7].

RefactorErl checks the correctness of transformations using complex static source code analysis and accurate transformations. On the other hand, due to the semantics of dynamic languages such as Erlang, the accuracy of checking the side-conditions by static analysis is limited, which means a regression test is needed even for refactorings.

Erlang applications are often tested with the property based testing tool QuickCheck [1]: the tool checks some properties given by the developers with random generated test inputs. Therefore we want to select those QuickCheck properties that should be retested after the source code is modified.

## References

[1] *Quviq QuickCheck*, 2013. URL *http://www.quviq.com/*.

[2] I. Bozó, D. Horpácsi, Z. Horváth, R. Kitlei, J. Kőszegi, M. Tejfel, and M. Tóth. RefactorErl – Source Code Analysis and Refactoring in Erlang. In *In proceeding of the 12th Symposium on Programming Languages and Software Tools, Tallin, Estonia*, 2011.

[3] *Refactorerl Homepage and Tool Manual.* ELTE-IK, PNYF, 2013. URL *https://plc.inf.elte.hu/erlang.*

[4] *Erlang Reference Manual.* Ericsson AB, 2013. URL *http://www.erlang.org/doc/reference_manual/ users_guide.html.*

[5] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts. *Refactoring: Improving the Design of Existing Code.* Addison-Wesley, 1999.

[6] S. Horwitz, T. Reps, and D. Binkley. Interprocedural slicing using dependence graphs. In *PhD thesis, University of Michigan, Ann Arbor, MI*, 1979.

[7] M. Tóth, I. Bozó, and Z. Horváth. Applying the query language to support program comprehension. In *Proceeding of International Scientific Conference on Computer Science and Engineering*, pages 52–59, Stara Lubovna, Slovakia, Sep 2010.

[8] M. Weiser. Program slices: Formal, psychological, and practical investigations of an automatic program abstraction method. In *ACM Transactions on Programming Languages and Systems, 12(1):3546*, Jan 1990.