Comparison of Programmers' Opinion in Change Impact Analysis

Gabriella Tóth

Change impact analysis is generally seen as a very difficult program comprehension problem. One of the reasons of this difficulty is that there is no universal definition for dependency between software artifacts, only algorithms that approximate the dependencies.

In the meantime, different kinds of such algorithms have been developed by researchers. But which algorithm is the most suitable in a specific situation, which one finds the relevant dependencies in the best way? Finding the most relevant dependencies is difficult, and is essentially a creative mental task.

A possible way to answer this important question is to involve programmers, and hear their subjective opinions based on expertise and experience in program comprehension. In this paper, we present such an experiment. We wanted to know what is the difference between not only some well-known algorithms and programmer's opinion, but between programmers' opinions as well, and hence we conducted a case study. This case study was documented earlier when the focus was on static impact analysis algorithms, not on programmers' opinion.

In this work, we report on our experiment conducted with this goal in mind using a compact, easily comprehensible Java experimental software system, simulated program changes, and a group of programmers (developers, computer science student and PhD students) who were asked to perform impact analysis with the help of different tools and on the basis of their programming experience. We applied several well-known algorithms (callgraph, program slicing, static execute after, historical co-change), JRipples[1], a Java framework for change impact analysis embedded in Eclipse development environment, and used BEFRIEND[2] to evaluate the results given by the programmers.

Now the author shows to which algorithms turned out to be the closest the individual programmer's opinion. Furthermore, the programmers are compared to each other according to their qualification, experiment, and the kind of dependencies identified by them.

References

[1] J. Buckner, J. Buchta, M. Petrenko, and V. Rajlich: JRipples: A tool for program comprehension during incremental change. IWPC, 2005.

[2] L. J. Fülöp, P. Hegedűs, and R. Ferenc: BEFRIEND - a Benchmark for Evaluating Reverse Engineering Tools, accepted, to appear. Periodica Polytechnica, 2009.