David Schuler: **Assessing Test Quality**

Promotion: Universität des Saarlandes, Naturwissenschaftlich-Technische Fakultät für Mathematik und Informatik
Erstgutachter: Prof. Dr. Andreas Zeller, Universität des Saarlandes
Zweitgutachter: Prof. Dr. Sebastian Hack, Universität des Saarlandes
Datum der Prüfung: 29. September 2011

Kurzfassung:

When developing tests, one is interested in creating tests of good quality that thoroughly test the program. This work shows how to assess test quality through mutation testing with impact metrics, and through checked coverage. Although there are different aspects that contribute to a test's quality, the most important factor is its ability to reveal defects, because software testing is usually carried out with the aim to detect defects. For this purpose, a test has to provide inputs that execute the defective code under such conditions that it causes an infection. This infection has to propagate and result in a failure, which can be detected by a check of the test. In the past, the aspect of test input quality has been extensively studied while the quality of checks has received less attention. The traditional way of assessing the quality of a test suite's checks is mutation testing. Mutation testing seeds artificial defects (mutations) into a program, and checks whether the tests detect them. While this technique effectively assesses the quality of checks, it also has two drawbacks. First, it places a huge demand on computing resources. Second, equivalent mutants, which are mutants that are semantically equivalent to the original program, dilute the quality of the results. In this work, we address both of these issues. We present the Javalanche framework that applies several optimizations to enable automated and efficient mutation testing for real-life programs. Furthermore, we address the problem of equivalent mutants by introducing impact metrics to detect non-equivalent mutants. Impact metrics compare properties of tests suite runs on the original program with runs on mutated versions, and are based on abstractions over program runs such as dynamic invariants, covered statements, and return values. The intention of these metrics is that mutations that have a graver influence on the program run are more likely to be non-equivalent. Moreover, we introduce checked coverage, an alternative approach to measure the quality of a test suite's checks. Checked coverage determines the parts of the code that were not only executed, but that actually contribute to the results checked by the test suite, by computing dynamic backward slices from all explicit checks of the test suite.