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.
The modernisation of existing software systems is an important topic in software engineering research and practice. A part of the modernisation of software systems is the restructuring of their architecture. This has to be done in numerous contexts, including the evolution to service-oriented architectures, the re-establishment of the maintainability of a system or the smooth migration of a system to a new development environment. Architecture restructurings are coarse-grained changes to the internal structure of the system that are performed in temporally limited projects. The planning of the transfer of an existing implementation to the target architecture of a system is currently a mostly manual task. While the analysis of the existing system is supported by e.g. architecture reconstruction approaches, the actual restructuring process is not supported by current approaches.

The MARE approach, which is introduced in this thesis, was developed to provide support for the stepwise restructuring of the implementation towards a target architecture. MARE supports architecture restructurings by semi-automatically creating a complete mapping of elements of the existing implementation to components of the target architecture. The creation of the mapping bases on explicit knowledge about the target architecture and its decomposition criteria. MARE employs graph clustering to implement the creation of the complete mapping.

The MARE method describes an iterative process model for the overall architecture restructuring process. It emphasises the target architecture as the basis for the architecture restructuring. The iterations of the process model allow for a stepwise restructuring of the system and the integration of human influence on the result of MARE.

The clustering algorithm employed by MARE to create the complete mapping bases on agglomerative hierarchical clustering. It is adjusted to incorporate knowledge about the target architecture. The decomposition criteria are considered by the definition of weights for the different types of dependencies that relate the elements of the existing implementation.

The MARE approach was evaluated in three case studies. These examined the application of MARE in small and middle-sized open source projects as well as for an industrial system with 3.5 million lines of code. The main goal of the evaluation is to show the quality and stability of the clustering algorithm. It furthermore shows the influence factors for the creation of the complete mapping.