Reengineering of Legacy Test Cases: Problem Domain & Scenarios

Ivan Jovanovikj, Marvin Grieger, Baris Güldali, Alexander Teetz
s-lab – Software Quality Lab, Paderborn University
Zukunftsmeile 1, 33102 Paderborn
{ijovanovikj, mgrieger, bguldali}@s-lab.upb.de, alexander.teetz@upb.de

1 Introduction

The constantly changing conditions, either business-driven or legal-driven, often lead to technological changes which causes a change of the existing software systems. According to [3], there are three possibilities of handling the changed conditions: to use a standard software, to develop a new system or to migrate an existing system. In this paper, we focus on the third option. Software migration is a process of transferring software systems into new environments without changing their functionality. In other words, it must be ensured that the migrated system behaves like the legacy system. A process that evaluates whether a system behaves in the intended way, both from functional and non-functional perspective, is software testing. In case of software migration, a set of test cases which are used to validate the existent system could already exist.

Software migration is established to reuse legacy systems. Therefore, we want to reuse test cases as well, especially since we have thousands of them. However, during a migration scenario, two causes need to be considered in the reuse of test cases: On the one hand, the way in which the system is changed implies a change of the test cases e.g. if the programming language is changed. On the other hand, the test environment may be exchanged in parallel due to test management improvement requirement. This can require restructuring of test cases. All in all, reusing test cases in a migration scenario is a challenging task.

In order to face this challenge, we envision a solution in the direction of coevolution of test cases. The idea is to provide a test cases reengineering method which can reflect the changes from the migrated system to the test cases as well as to modify them so they can be reused in the new test tool.

In this paper, we first describe the problem domain of reengineering test cases. Thereafter, we present two general scenarios. Next, we present an observation of an industrial context where these scenarios were initially identified. We conclude our work with a sketch of possible solution direction and we raise some open questions regarding the challenges related to the envisioned solution.

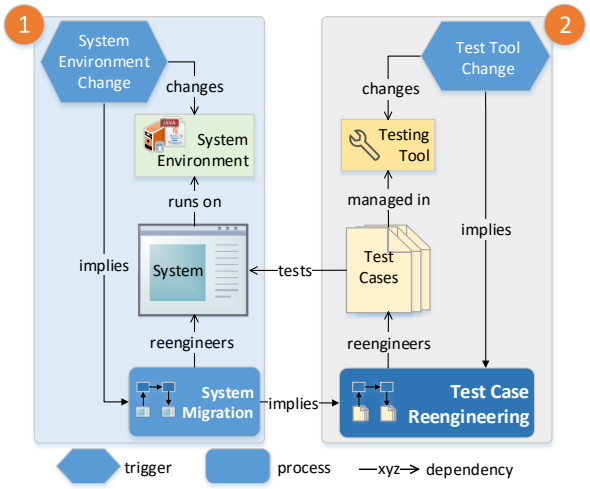

Figure 1: Problem domain and possible scenarios in a migration project

2 Problem Domain of Test Case Reengineering

In a software migration scenario, two causes can influence the reengineering of test cases: system environment change or test tool change. When the system environment changes, it triggers a migration of the system to the new environment (left side of Figure 1). On technical level, the migration is performed by enacting a transformation method that is an instance of the well-known horseshoe model [1]. It includes the following processes: reverse engineering, restructuring and forward engineering. Reverse engineering is "the process of analyzing a subject system to create representations of the system in another form or at a higher level of abstraction." [1]. Restructuring is "the transformation from one representation form to another at the same relative abstraction level, while preserving the subject system's external behavior" [1]. Finally, forward engineering is "the traditional process of moving from high-level abstractions and logical, implementation-independent designs to the physical implementation of a system" [1].

Since changes occur during the restructuring process, software migration implies test case reengineer-

ing, whose goal would be to enable appropriate set of test cases for the migrated system. It should take into account the changes from the restructuring process in the software migration. Test cases, as the central artifact in the testing process, describe sequence of conditions that have to be validated. Depending on the used migration strategy (re-implementation, conversion and encapsulation [3]) and the testing level (unit test, integration test, system test), the existing test cases are affected differently and need to be modified in appropriate way, as mentioned in [2]. The changes that appear here are of a semantic nature, i.e., the conditions or the test data of the test cases need to be modified.

Test case reengineering could be implied also by a decision to change the testing tool (right side of Figure 1). A test tool is the software that is used in one or more testing activities. For example, it is used for analysis, design and implementation of test cases. Test cases have a structure which is dependent on the testing tool characteristics. A change of the testing tool would mean that the test cases would have to be reengineered in order to match the characteristics of the target test tool, so they would have to undergo structural (syntactical) changes.

In the following section, we present an exemplary scenario as observed in practice.

3 Exemplary Scenario

In an industrial context, we observed a project that dealt with the reengineering of test cases. Thereby, both causes were present, i.e., the system environment as well as the test tools were exchanged. Figure 1 shows this particular scenario.

As a result, the reengineering of the test cases was performed as follows: first, the change of the test tool was addressed by defining a corresponding reengineering method, whereas the change of the system environment needs to be addressed, it is still ongoing work. The goal of the reengineering method was to migrate the test cases in a way that improves the test coverage, the structuredness and the maintenance of test cases. However, dealing with thousands of test cases without a structure, with existing erroneous and obsolete test cases was not an easy task. The transformation method was automized up to some extent, using transformation scripts, but still with a considerable manual effort. Additionally, the transformation was done in isolation, i.e., without migrating the software system.

By our research, we aim to address the open problems of this scenario, i.e., the reengineering of the test cases based on the migration of the system, setting the needed manual effort to minimum. In that sense, we aim to provide a solution concept to coevolve the test cases. We sketch the idea in the next section.

4 Solution Idea

After we have seen the possible scenarios and the open challenges, in the following we briefly discuss our solution idea for test case reengineering, i.e., how to enable coevolution of test cases.

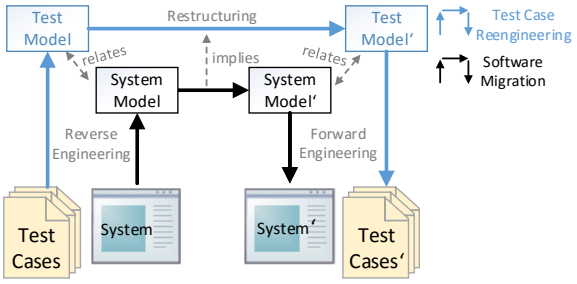

Figure 2: Coevolution of test cases

Similarly as in the case of software migration, test cases are reengineered by following the horseshoe model, as shown in Figure 2. Since our focus is on model-driven software migration, our envisioned solution goes in the direction of model-based testing. First, using extraction techniques, a test model is extracted from the existing test case. Then, the changes from the artifacts in software migration on the same abstraction level as the test model, are propagated to the test model. In our opinion, propagating changes from artifacts on the same abstraction level would be easier than propagating them from higher abstraction level directly to the existing test cases. The last step would be to use the modified model to generate the actual test cases for the migrated system. To the best of our current knowledge, no work has been done in this direction.

This solution idea raises some questions. First of all, what level of abstraction would be most suitable to relate the models and reflect the changes from the software migration horseshoe to the test case reengineering horseshoe? Then, what are the limitations for automation of this process? Would it be suitable for all types of test cases? Also, could the software migration benefit from the extraction of test models? We envision to answer these questions by future research.

References

[1] E. J. Chikofsky and J. H. Cross II. Reverse engineering and design recovery: A taxonomy. *IEEE Software*, 1990.

[2] M. Grieger, B. Güldali, S. Sauer, and M. Mlynarski. Testen bei migrationsprojekten. *OBJEKTspektrum*, September 2013.

[3] H. M. Sneed, E. Wolf, and H. Heilmann. *Software-Migration in der Praxis: Übertragung alter Softwaresysteme in eine moderne Umgebung*. dpunkt Verlag, 2010.