Towards Applying Model-based Testing in Test Case Migration

Ivan Jovanovikj[1], Baris Güldali[2], Marvin Grieger[1]
[1]s-lab – Software Quality Lab, Paderborn University
[2]S&N CQM Consulting & Services GmbH, Paderborn
Email: {ijovanovikj, mgrieger}@s-lab.upb.de, baris.gueldali@sn-cqm.de

1 Introduction

After performing migration of a software system, it must be proved that the migrated system still behaves like the legacy system regarding the functionality. As a technique to validate the migrated system, software testing is being used. According to [1], testing in software migration projects is a costly endeavor. For that reason, when existing test cases are available, their reuse should be considered. Two main things motivate their reuse: costs save and knowledge reuse. They come from the fact that the functionality of the system after the migration must stay unchanged. Hence, the existing test cases are valuable source of information about the functionality of the legacy system and consequently for the migrated system as well.

Reusing test cases in a software migration scenario introduces several challenges. For example, the quality of the test cases needs to be assessed, because they might have become legacy due to the evolutionary development. The existing test case set may contain duplicate, obsolete or even erroneous test cases. Another example is the changes that happen to the system (language, framework or architectural change) may influence the existing test cases [2]. Reflecting these changes to the test cases is an important prerequisite in enabling a test case set that could validate the migrated system. Dealing with such changes on the level of test cases could be difficult and therefore we assume that abstracting from test cases and extracting test models, could ease this task.

Motivated by the idea of model-driven software migration, we envision a novel test case migration method that combines reverse engineering and model-based testing. Model-based testing(MBT) is a testing technique that relies on abstract models, which are used to derive test cases for an implementation [3]. In the literature, several different scenarios have been introduced depending on the origin of the test model [3, 4]. From our perspective, the scenario where the test models are reverse engineered from existing test cases (known as '*Models from Test Cases*' in [4]) is the most interesting one.

Following the model-driven software migration horseshoe[1], we first extract test models out of the test cases, then we restructure the test models by reflecting the changes from the software migration, and finally, we generate test cases for the migrated system. On the one hand, it could be seen as a test case re-engineering method that enables migration and co-evolution of test cases. On the other hand, from the model-based testing perspective, it could be seen as a new scenario which is an extension of the previously mentioned scenario '*Models from Test Cases*'.

In this paper, we first discuss the field of model-based testing and reverse engineering and thereafter we sketch our method and discuss how it extends the existing model-based scenario. At the end, we discuss the requirements that model-based testing imposes on our method.

2 Background

Model-based testing is a software testing technique which derives (manually or automatically) test cases from models that describe the desired functionality of the system under test. Regarding the origin of the models, six different scenarios have been identified so far [3, 4]. In the context of test case migration, most interesting scenario is the scenario '*Models from Test Cases*' where the test models are reverse engineered from the existing test cases.

This scenario assumes that an existing set of test cases contains relevant information about the system under test as well as about the test inputs and expected results. The existing test cases are basically reused and by using reverse engineering techniques, the test models are automatically derived from these test cases. This scenario is described as useful in case of migration from classical to model-based testing [4]. After applying reverse engineering, the obtained test models could be used for generation of new test cases.

Reverse Engineering is the the enabling part in the previously mentioned scenario. According to [5], 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". In

[1]adm.omg.org Architecture-Driven Modernization

the model-based testing context, the test cases are the artifacts being reverse engineered and the result of the reverse engineering process are the test models. Existing reverse engineering techniques for test cases are based on machine learning algorithms [6, 7], parallel composition (synthezis) [8], process mining [9] etc.

3 Solution Idea

Aiming to reuse test cases in software migration scenario, we envisioned a solution that employs model-based testing. Generally, it represents an extension of the previously discussed scenario and could be seen as a new model-based scenario in the context of software migration.

Our method, as shown in Fig. 1, consists of six steps: 1) *quality assessment* of the existing tests cases; 2) *reverse engineering* of the test cases; 3) establishment of *relation* between the test model and system model; 4)*refactoring* of the test models; 5) *restructuring* of the test models; 6) *forward engineering* of test cases i.e. the actual test case generation; 7) *quality assessment* of test cases, i.e., the validation of the migration of test cases.

The first step, the quality assessment of test cases, is performed to decide whether eventual reuse of test cases would be beneficial. Based on a quality model for test cases and appropriate metrics [10], a check of quality characteristics of test cases is performed. Quality characteristics like effectiveness, understandability, structuredness, traceability between test cases and system or code components or test coverage are checked. This step could be seen as an extension of the model-based testing scenario.

The second step is the reverse engineering of the test cases. Since any test level may be needed during migration of test cases, the reverse engineering phase has to provide support for all test levels. Furthermore, the selected techniques have to be able to cope with non-automated and in natural language described test cases (mainly system tests). For these purposes, existing reverse engineering techniques from model-based testing could be re-used [8, 6, 7, 9], or, in case of specific requirements (non-automated test cases, different levels of granularity), a new technique may be developed. Fig. 2 shows an example of how integration tests can be represented using a component diagram, whereas Fig. 3 show an example of a test model of system tests using an activity diagram.

After this step, we establish a relation between the reverse engineered test models and the models of the system in order to discover eventual inconsistencies between the models as well as to enable later reflection of relevant system changes to our test models. Establishing relationships between the test models and system models is a challenging task since different modeling notations may be used, or the models may be at different levels of abstraction. Fig. 2 shows an example how a test model and a model of the system can be related. The relations(represented as blue dashed lines) are established between corresponding elements in the model of the system and the test models.

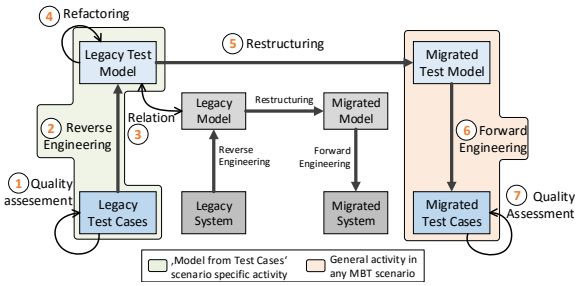

Figure 1: Model-driven method for migrating test cases

Once the relation between the test models and the models of the system is established, refactoring is performed. We analyze our test models to identify eventual inconsistencies with the models of the system, which could be a consequence of obsolete or erroneous test cases. Due to the evolutionary development of the system as well as the test cases, it may happen that non-existing interfaces (integration testing, Fig. 2) or paths in the system (system testing, Fig. 3) are being tested. Based on the previously established relations, potential inconsistencies can be discovered.

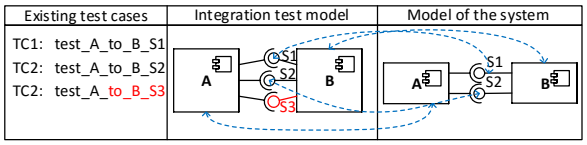

Figure 2: Relating an integration test model and a model of a system

The next step is the step of restructuring of the test models. This step is actually a consequence of the changes that happen in the system during its restructuring phase. Since these changes may be relevant for the test models, our idea is to use the previously established relation between the models of the system and the test models in order to do a reflection of the relevant changes. Let us assume that the model of the system in Fig. 2 initially was same as the integration test model, i.e., the system was consisted of the components *A* and *B*, and the interfaces *S1*, *S2* and *S3*. The system was so restructured that the interface *S3* was removed. After it was restructured, it looks as presented in Fig. 2. This change should be reflected on the integration test model as well, since the interface *S3* does not have to be tested anymore.

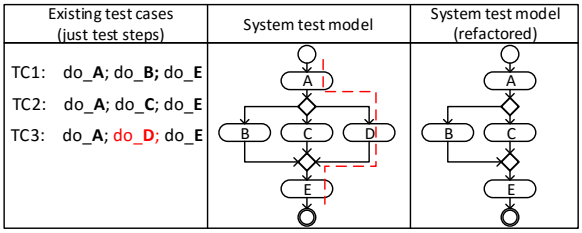

Figure 3: Detection of an obsolete test case

The last step is the the step of forward engineering of test cases where the restructured test models are actually used for test case generation. At the very end, a quality assessment is performed in order to validate the migration of the test cases. For example, a test coverage is checked and it should be at least as high as it was before the migration. Similarly, other quality attributes from the quality model [10] may be checked as well.

4 Discussion

In the previous section, going step by step through our method, we have shown how our approach extends the existing model-based scenario. Therefore, the overall method could be seen as a new model-based scenario in the context of software migration. Due to the specifics of the context, it has some additional activities like quality assessment, relation, refactoring and resructuring based on reflection of the system changes to the test cases.

Additionally, model-based testing imposes some specific requirements as well. As mentioned in [3], a big concern in model-based testing is the abstraction level of the test models, i.e., should the test models be more abstract or with more details. This requirement would directly influence the reverse engineering activity of our method.

According to the taxonomy presented in [11], several other requirements could be derived regarding the test model, test generation and test execution. The redundancy dimension in the test models is very important. Since the origin of the models in our case are the existing test cases, which we assume that in the case of legacy system are designed independently in a conventional way, we ensure high redundancy compared to the development models. Furthermore, the reverse engineered test models could be used as a detailed specification of the old system and be useful in the migration process. Another requirement is regarding the model characteristics, namely the nondeterminism of the models. The modeling paradigm used to describe the test models has also direct influence on the reverse engineering of test models.

The requirements regarding the test generation, influence the forward engineering activity of our method. First, the test coverage has to be specified on the reverse engineered and restructured test models as well as the technique used for test case generation. In the case of software migration, high automation of test case generation is desired. The last requirement is about the test execution, which could be either on-line or off-line. In principle, regression testing is used in software migration scenarios. Therefore, the offline scenario would be the more suitable one.

We believe that our initial idea is a good starting point in providing a test case migration framework which is applicable in different contexts, i.e., different types of system migration projects. The proposed model-driven method is still on a conceptual level and our further steps would be to concertize each of the steps by applying concrete methods and techniques. Once concretized, the method can be evaluated against a real set of test cases in a real software migration context.

References

[1] H. M. Sneed, "Risks involved in reengineering projects," 1999.

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

[3] A. Pretschner et al., "Methodological issues in model-based testing," 2004.

[4] B. Güldali et al., "Effort comparison of model-based testing scenarios," 2010.

[5] E. Chikofsky et al., "Reverse engineering and design recovery: A taxonomy," *IEEE Software*, 1990.

[6] E. Werner and J. Grabowski, "Model Reconstruction: Mining Test Cases," 2011.

[7] H. Hungar, T. Margaria, and B. Steffen, "Test-based model generation for legacy systems," 2003.

[8] A. Jääskeläinen et al., "Synthesizing Test Models from Test Cases," 2009.

[9] D. Xu, W. Xu, B. K. Bavikati, and W. E. Wong, "Mining executable specifications of web applications from selenium ide tests.," 2012.

[10] H. M. Sneed, "Measuring the effectiveness of software testing," 2004.

[11] M. Utting et al., "A taxonomy of model-based testing approaches," 2012.