Towards a Framework for Constructing Context-Specific Migration Methods for Test Cases

Ivan Jovanovikj, Stefan Sauer
s-lab – Software Quality Lab, Paderborn University
Zukunftsmeile 1, 33102 Paderborn
{ivan.jovanovikj, sauer}@s-lab.uni-paderborn.de

Abstract

Test case reuse in migration projects brings double benefit: reuse of valuable knowledge as well as time and cost savings. Due to system changes, a direct reuse of test cases might be impossible. To facilitate reuse, the migration context, from both system migration as well as testing perspective, has to be considered. For example, system changes need to be detected, understood and then reflected to the test cases or the characteristics of the target testing framework have to be identified. In this paper, we present a novel framework that enables construction of context-specific migration methods for test cases.

1 Introduction

Software migration is a process of transferring software systems into new environments without changing their functionality. In legacy migration, for example, a legacy system still having some value is migrated into a new environment. Another use case is when a system is migrated to another platform, thus a multi-platform provision is enabled (often seen in mobile app development today). Software testing is an important activity in software migration as it verifies whether the migrated system still provides the same functionality as the source system. Since software migration is established to reuse existing systems, we want to reuse test cases as well. The reuse of test cases can be beneficial, not just from economical perspective, but also from practical perspective: the existing test cases contain valuable information about the functionality of the source system.

However, in some migration scenarios a direct reuse of test cases might be impossible due to system changes. Since the test cases are coupled with the system they are testing, the system changes need to be detected, understood and then reflected on the test cases to facilitate reuse. In other words, the test cases need to be co-evolved. However, co-evolving test cases is far from being trivial since several challenges need to be addressed [5], like quality assessment, refactoring or reflection of system changes.

In this paper, we present a novel framework for the construction of context-specific migration methods for test cases. The resulting migration methods enable automated co-evolution of test cases for a specific migration context. Firstly, the system migration and testing contexts are characterized. Then, based on the context information, a reference migration method gets tailored. Following the idea of model-driven software migration, the reference migration method extracts a test model out of the existing test cases, reflects the system changes with the help of the context model, and at the end, in a model-based testing manner, generates test cases for the migrated system. We present a case study from an industrial project in which a migration of an existing modeling framework from Java to C# was performed.

2 Construction of Context-Specific Migration Methods for Test Cases from a Reference Method

Situational method engineering (SME) deals with the construction of methods which are adapted to a specific situation. Regarding the degree of flexibility, different SME approaches exist [3]. Our approach relies on the techniques of method tailoring. As shown in Fig. 1, the overall approach consists of three main phases: *Pre-Migration*, *Migration*, and *Post-Migration*.

In *Pre-Migration*, the migration context is analyzed from both testing perspective and system migration perspective. This initial analysis of the context shall answer the question whether eventual reuse of test cases would be beneficial or not. If yes, the results of the context analysis are used in the migration phase, namely in the adaptation of the reference migration method.

Migration is the main phase which actually performs the migration itself. Motivated by the work presented in [3], we rely on method tailoring to enable development of context-specific test case migration methods which support co-evolution of test cases.

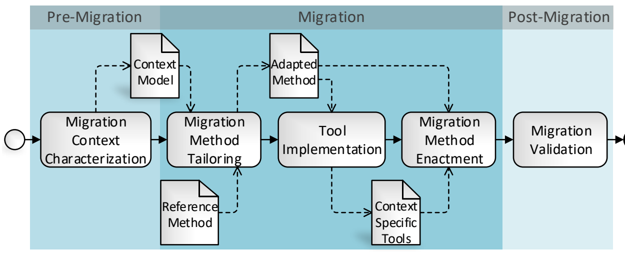

Figure 1: Approach Overview.

The *Migration* phase involves the following activities: method tailoring, method implementation, and method enactment. Having the context information collected, the reference migration method gets tailored. Through a sequence of adaptations, like modification, deletion or addition of actions and/or artifacts, the reference method gets adapted so that it is suitably applicable in the particular context. Hence, the resulting adapted method may not contain all of the steps or some of the steps may be modified in a specific way with regards to the context characteristics. Then, in the implementation step, the situation-specific toolchain is developed. At the end, the enactment of the developed migration method takes place.

In *Post-Migration*, the migration of the test cases needs to be validated. This validation involves checking whether certain requirements have been met, e.g., some test quality criteria like test coverage.

3 Reference Migration Method

Following the idea of model-driven software migration, our reference migration method shown in Fig. 2 consists of the following activities: *Reverse Engineering*, *Restructuring*, and *Forward Engineering* [2]. On a technical level, the reference method is an instance of the well-known horseshoe model. Our method includes also an additional activity, *Refactoring*.

Reverse Engineering can be seen as a combination of *Model Discovery* and *Model Understanding* [1]. *Model Discovery* is an automatic text-to-model transformation activity which results in a model of the test case source code. *Model Understanding* is a model-to-model transformation activity, which takes a platform-specific model and transforms it to a platform-independent test model. This transformation performs a semantic mapping and results in a test model of higher level of abstraction.

Then, in *Refactoring*, we analyze our test model to identify eventual inconsistencies with the model of the system, which could be a consequence of obsolete or erroneous test cases.

Restructuring comes as a consequence from the changes that happen in the system during its restructuring phase as well as the changes in the target testing environment, e.g., new testing framework.

During *Forward Engineering*, the restructured platform-independent test model is firstly refined to a platform-specific test model applying a model-to-model transformation. This step is called *Test Case Concretization*. In *Code Generation*, the test model is further used as input for the model-to-text transformation which at the end generates the test code into the desired target environment [4]. As shown in Fig. 2, the last two steps can be combined in one.

4 Preliminary Results

Our method was applied in an industrial project which main goal was to migrate parts of the well known Eclipse Modeling Framework (EMF) along with the

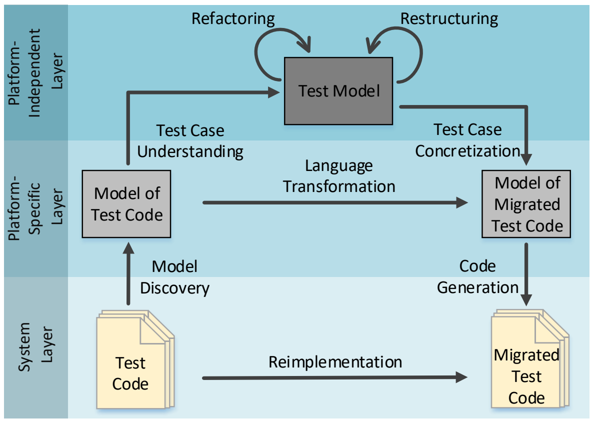

Figure 2: Reference Migration Method.

Object Constraint Language (OCL) from Java to C#. EMF and OCL are stable and well-tested and all test cases are available on public code repositories. Our goal in this particular case study was to reuse the OCL test cases in order to validate the OCL functionality in the target environment. All in all, 13 different test suites were processed, each of them addressing different functional aspects of OCL. The source environment was JUnit and MS Unit Test Framework was selected as a target testing environment for the test cases. The main change that has been performed in the system migration was the change from just-in-time (JIT) compilation to ahead-of-time (AOT) compilation. Since the test cases were implemented in JIT-fashion, a change to AOT was necessary. On the base of this context information, the reference migration method was adapted. The adapted method included *Model Discovery* and *Test Case Understanding*, which resulted in a platform-independent test model. Then, a direct *Code Generation* was selected as a last step. Corresponding tooling in terms of parsers and generators was developed and the method was enacted. The overall result was automated migration of over 92% out of 4000 existing test cases.

References

[1] H. Bruneliere, J. Cabot, F. Jouault, and F. Madiot. Modisco: A generic and extensible framework for model driven reverse engineering. 2010.

[2] E. J. Chikofsky and J. H. Cross II. Reverse engineering and design recovery: A taxonomy. 1990.

[3] M. Grieger, M. Fazal-Baqaie, G. Engels, and M. Klenke. Concept-based engineering of situation-specific migration methods. 2016.

[4] A. Z. Javed, P. A. Strooper, and G. N. Watson. Automated generation of test cases using model-driven architecture. 2007.

[5] I. Jovanovikj, M. Grieger, and E. Yigitbas. Towards a model-driven method for reusing test cases in software migration projects. 2016.