Model-Driven Co-Migration of Test Cases

Ivan Jovanovikj
Paderborn University
Fürstenallee 11, 33102 Paderborn, Germany
{ivan.jovanovikj}@upb.de

1 Introduction

Software testing plays an important role in software migration as it verifies whether the migrated system still provides the same functionality as the original system which is the main requirement in a software migration. As 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 and therefore, about the functionality of the migrated system, too. Before starting with the actual migration, a decision is necessary whether it is beneficial to perform a migration of the existing test cases, i.e., the quality of the test cases has to be evaluated. If yes, then a test case migration takes place and it is performed by development and enactment of a test migration method. A test migration method defines which activities should be performed, which tools to be applied, and what artifacts should be generated in order to transfer a system from one environment to another. Another aspect is the dependency between the system and the test cases. As 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, i.e., 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, like quality assessment, refactoring or reflection of system changes. For this reason, we need a method which is capable of capturing such information, and accordingly enabling the development of a suitable test case migration method. Last but not least, it should be assured that one can have confidence in the migrated test cases when executing them against the migrated system.

2 Challenges

Based on the discussion in the previous section, providing an end-to-end solution for test case migration is quite challenging. In the following, we discuss the general challenges related to test case migration.

C1: How to make a systematic quality evaluation of test cases?

Reusing test cases is beneficial, only when the test cases have some value. Migrating test cases that are redundant or do cover parts of the system that are not used anymore or if they have a low quality, e.g., if their structuredness or effectiveness is low, can result in migrated test cases which cannot properly validate the migrated system. Therefore, before doing anything with the test cases, their quality needs to be evaluated.

C2: How to incorporate the co-evolution analysis in test case migration?

The migration of the test cases is coupled with the migration of the system they are testing. In other words, reusing test cases implies a co-migration together with the system. Therefore, the system changes need to be detected, and then their impact on the test cases analyzed, and if necessary propagated.

C3: How to transform the test cases in an automated way?

When migrating a large number of test cases, the test case migration should be ideally completely automated. The existing test cases may be structurally complex and as a result a direct transformation would be hardly possible. All in all, the solution approach should support the typical reengineering activities like reverse engineering, restructuring, and forward engineering for test cases.

C4: How to provide a situation-specific test case transformation method?

As each migration project is performed in a specific migration scenario, a "one-size-fits-all" test case migration method is not a viable solution. In general, the test case changes depend on the system changes as well as on the requirements imposed by the new test environment. Not considering the complete situational context may result in test case transformation methods that are inefficient and/or ineffective.

C5: How to validate a test case migration?

As in software migration, the main requirement in test case migration is to transfer the test cases to a new environment without changing their "functionality", i.e., without changing the expected behavior the test cases assert. As the migrated test cases are used as safeguards for the system migration, their correct migration is crucial. But, as the test cases change along with the system, it is challenging to validate their migration.

3 Solution Idea

In order to address the aforementioned research question, i.e., the challenges that led to the research question, we propose a framework which provides an end-to-end-solution for the test case co-migration. The basic idea evolves around the double horseshoe reengineering model which we propose as a solution to the co-migration problem. Motivated by the idea of model-driven software migration, the solution approach applies the Model-driven Engineering (MDE) principles for the migration of test cases. Furthermore, it combines techniques from *Software Evolution* and *Situational Method Engineering (SME)* to address the co-evolution and situativity challenges, respectively. Thus, the resulting model-driven migration methods enable automated co-evolution of test cases for a specific migration scenario. Additionally, as shown at the bottom left-hand corner of Figure 1, *Test Case Quality Evaluation* provides means to check the quality of the existing test cases. At the bottom right-side corner of Figure 1, *Test Case Migration Validation* validates the test case migration. As shown in Figure 1, the test case migration framework supports the three general migration phases: *Pre-Migration Phase*, *Migration Phase*, and *Post-Migration Phase*. The labels from **C1** to **C5** represent the previously introduced challenges and show which part of the solution addresses which challenge.

Pre-Migration Phase. During this phase *Test Case Quality Evaluation* is performed to evaluate the quality of the existing test cases. Our approach enables a systematic and efficient development of quality plans (*Test Case Quality Plan Development*) which consider the context information and integrate a standardized quality model, namely the ISO/IEC 25010. A quality plan serves as a guideline for the quality evaluation of test cases and emphasizes the context of use of test cases as a major factor of influence for the whole quality evaluation. After a quality evaluation is performed, and an indication on the quality of the test cases is obtained, a decision is made whether the existing test cases should be migrated or not, i.e., whether to proceed to the migration phase or not. **Tool.** This phase is supported by a tool for the creation of quality plans.

Migration Phase. During the main phase, two general steps need to be taken: *Method Development* and *Method Enactment*. Following the basic idea of the existing *Method Engineering Framework for Situation-Specific Software Transformation Methods (MEFiSTo)*, during *Method Development* and *Method Enactment*, a situation-specific test case migration method is developed and enacted, respectively. The general idea is to use predefined method blocks, called *Method Fragments* which are stored in a *Method Base*. A method fragment is an atomic

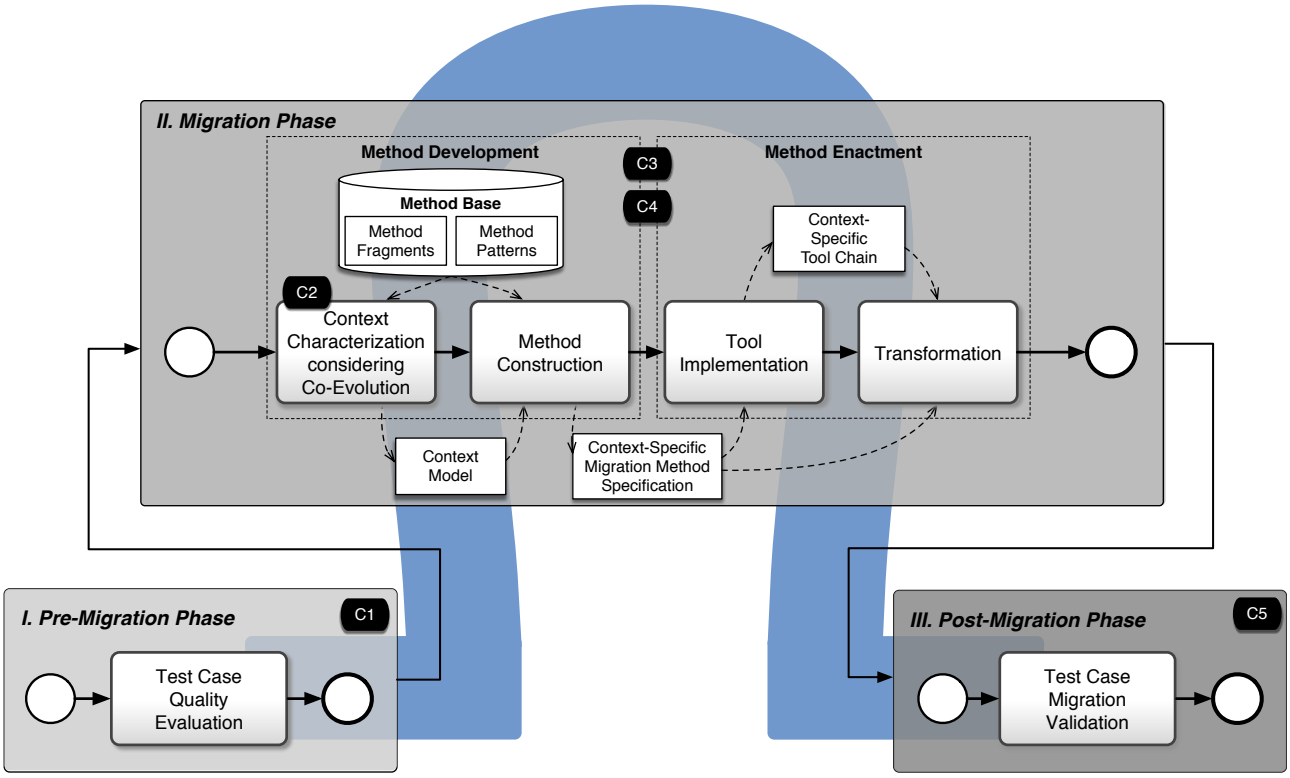

Figure 1: The general phases with the corresponding activities of our approach

building block of a migration method, i.e., an activity, artifact or tool. *Method Patterns*, on the other hand, represent a proven migration strategy and show how different migration fragments could be combined to realize this strategy. Each pattern has a set of characteristics which actually express its suitability to a certain situation. Having the method fragments and method patterns, a guidance is needed on how to create a migration method for a specific context. This is done by a method engineering process, which guides the development and the enactment of the context-specific migration method. The method engineering process begins with *Context Characterization considering Co-Evolution* which takes the original test code as a main input. During this activity, both the system migration and testing contexts are being characterized. As part of this activity, the co-evolution analysis is performed, namely change detection and impact analysis. The overall outcome of the analysis, namely the *Impact Model*, which is part of the *Context Model*, captures the impact the system changes have on the test cases. Then, based on the previously collected context information in terms of the *Context Model*, an appropriate method pattern is selected and configured. The result of this activity, *Context-Specific Migration Method Specification*, is used as a base input for the *Tool Implementation*, where for every specified activity that shall be performed (semi-)automatically, an appropriate tool is implemented. The migration phase ends with the *Transformation* step, where on the basis of the *Context-Specific Migration Method Specification* and the *Context-Specific Tool Chain* the actual transformation of the test cases is performed. **Tool.** This phase is supported by a modeling tool for the creation of test case co-migration methods.

Post-Migration Phase. Last but not least, in the *Post-Migration Phase* the migrated test cases are validated. With the help of a novel validation method, it is checked whether the test cases are migrated without changing their behavior, i.e., without changing what they actually test. As the main goal of the migration validation is to identify false positives and false negatives among the migrated test cases, the validation method relies on mutation analysis. **Tool.** This phase is supported by a mutation framework which supports the mutation analysis.

4 Publication Overview

In the course of our work, a number of publications were created and published on different workshops and conferences. As shown Figure 2, the classification is made according to the phase of the solution approach a given publication is related to.

5 Matching Requirements

As the solution idea is based on situational method engineering, it imposes no requirements on particular technology, language or platform. The only thing that can be seen as a kind of prerequisite is that beside the system to be migrated also test cases are provided.

References

[JEAS18] Ivan Jovanovikj, Gregor Engels, Anthony Anjorin, and Stefan Sauer. Model-driven test case migration: The test case reengineering horseshoe model. In *Information Systems in the Big Data Era - CAiSE Forum 2018, Tallinn, Estonia, June 11-15, 2018, Proceedings*, pages 133–147, 2018.

[JGG16] Ivan Jovanovikj, Baris Güldali, and Marvin Grieger. Towards applying model-based testing in test case migration. *Softwaretechnik-Trends*, 36(3), 2016.

[JGGT16] Ivan Jovanovikj, Marvin Grieger, Baris Güldali, and Alexander Teetz. Reengineering of legacy test cases: Problem domain & scenarios. *Softwaretechnik-Trends*, 36(3), 2016.

[JGY16] Ivan Jovanovikj, Marvin Grieger, and Enes Yigitbas. Towards a model-driven method for reusing test cases in software migration projects. *Softwaretechnik-Trends*, 36(2), 2016.

[JNES18] Ivan Jovanovikj, Vishwak Narasimhan, Gregor Engels, and Stefan Sauer. Context-specific quality evaluation of test cases. In *Proceedings of the 6th International Conference on Model-Driven Engineering and Software Development, MODELSWARD 2018, Funchal, Madeira - Portugal, January 22-24, 2018.*, pages 594–601, 2018.

[JNY+20] Ivan Jovanovikj, Achyuth Nagaraj, Enes Yigitbas, Anthony Anjorin, Stefan Sauer, and Gregor Engels. Validating test case migration via mutation analysis (to appear). In *Proceedings of the 2020 IEEE/ACM 1st International Conference on Automation of Software Test (AST 2020)*, 2020.

[JS17] Ivan Jovanovikj and Stefan Sauer. Towards a framework for constructing context-specific migration methods for test cases. *Softwaretechnik-Trends*, 37(2), 2017.

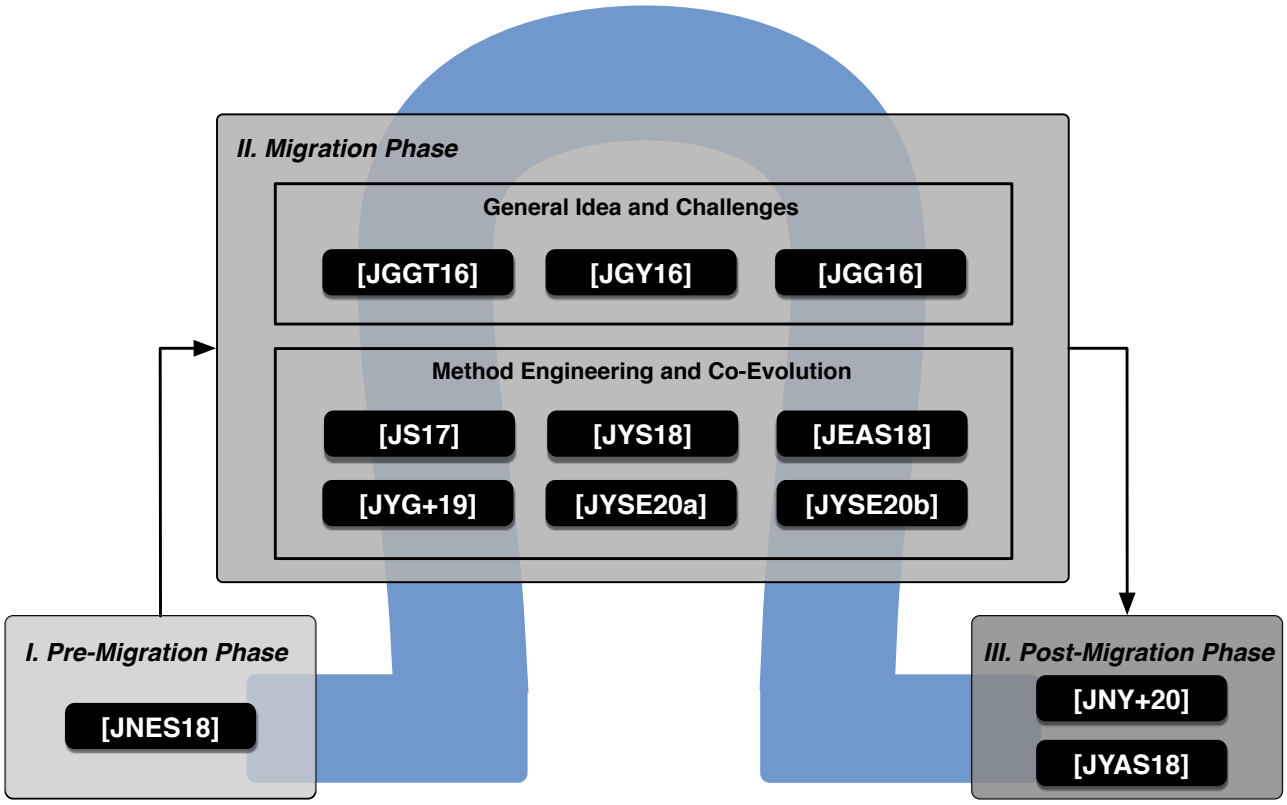

Figure 2: Overview of the publications

[JYAS18] Ivan Jovanovikj, Enes Yigitbas, Anthony Anjorin, and Stefan Sauer. Who guards the guards? on the validation of test case migration. *Softwaretechnik-Trends, Proceedings of the 20th Workshop Software-Reengineering & Evolution (WSRE) & 9th Workshop Design for Future (DFF)*, 2018.

[JYG+19] Ivan Jovanovikj, Enes Yigitbas, Marvin Grieger, Stefan Sauer, and Gregor Engels. Modular construction of context-specific test case migration methods. In *Proceedings of the 7th International Conference on Model-Driven Engineering and Software Development, MODELSWARD 2019, Prague, Czech Republic, February 20-22, 2019.*, pages 534–541, 2019.

[JYS18] Ivan Jovanovikj, Enes Yigitbas, and Stefan Sauer. Test case migration: A reference process model and its instantiation in an industrial context. In *Joint Proceedings of the Workshops at Modellierung 2018 co-located with Modellierung 2018, Braunschweig, Germany, February 21, 2018.*, pages 153–162, 2018.

[JYSE20a] Ivan Jovanovikj, Enes Yigitbas, Stefan Sauer, and Gregor Engels. Concept-based co-migration of test cases (to appear). In *Proceedings of the 8th International Conference on Model-Driven Engineering and Software Development - Volume 1: MODELSWARD*, 2020.

[JYSE20b] Ivan Jovanovikj, Enes Yigitbas, Stefan Sauer, and Gregor Engels. Test case co-migration method patterns (to appear). *Software Engineering 2020 Workshopband*, 2020.