A Method to Systematically Improve the Effectiveness and Efficiency of the Semi-Automatic Migration of Legacy Systems

Masud Fazal-Baqaie, Marvin Grieger, Stefan Sauer
Universität Paderborn, s-lab – Software Quality Lab
Zukunftsmeile 1, 33102 Paderborn
{mfazal-baqaie, mgrieger, sauer}@s-lab.upb.de

Markus Klenke
TEAM GmbH
Hermann-Löns-Straße 88, 33104 Paderborn
mke@team-pb.de

1 Introduction

Legacy systems, e.g. applications that have been developed using a 4th generation language (4GL), need to be modernized to current technologies and architectural styles in order to ensure their operation in the long run. In practice, a true modernization cannot be achieved by fully automated transformation. As a result, a custom migration tool chain transforms only parts of the legacy system automatically, while a manual completion of the generated source code is still necessary. Two different roles are responsible for these activities, carried out incrementally. A small group of *reengineers* conceptualizes and realizes the migration tool chain while a larger group of *software developers* completes the generated source code by reimplementing missing parts. Thus, the overall effectiveness and efficiency of the migration comes down to optimizing the generated source code as well as the instructions on how to manually complete it.

In this paper, we describe a method to systematically improve the generated source code and the corresponding instructions by exchanging structured feedback between developers and reengineers. We also summarize first experiences made with this method, which is currently applied in an industrial project [1].

2 Feedback-Enhanced Method

The Reference Migration Process (ReMiP) [2] provides a generic process model for software migration. It states that, first, the reengineers define a migration path, a corresponding tool chain as well as migration packages for the application. Then, the developers iteratively process the migration packages, completing the generated source code provided by the reengineers. Experiences they make during this completion can be used to improve the effectiveness and efficiency of iterations to follow. Communicating these experiences with the group of reengineers enables them to improve the generated source code, however, ReMiP does not describe such activities.

Our method refines ReMiP by describing when, by whom, and how feedback is collected and integrated during each iteration. Figure 1 shows the method modeled in SPEM [3]. It groups the activities performed by the software developers into the activity named *Development Activity* and the activities carried out by the reengineers into the activity named *Reengineering Activity*. In each iteration, first the developers carry out their activity and then the reengineers carry out theirs. During the *Development Activity*, the *Manual Reimplementation Instructions* and the *Generated Source Code* are taken as input, while the *Feedback Entry List* is generated as output. Conversely, the *Feedback Entry List* is an input for the *Reengineering Activity* where the reengineers adapt the instructions and the generated code according to the feedback. In the following sections, we will discuss some important aspects of the activities described.

2.1 Reflection during Development

As depicted in Figure 1, we extended the usual *Transformation & Test Activities* with a specific *Reflection* task that is performed in parallel to them and that has the *Feedback Entry List* as output. Simplified, in SPEM-Terminology a *task* is a precisely described unit of work while an *activity* contains a nested structure of various activities (and tasks). The intention of the *Reflection* task is to collect viable feedback that can be used by a reengineer to improve the *Generated Source Code* as well as *the Manual Reimplementation Instructions* for subsequent iterations of the migration. Focusing on the right information minimizes the effort to collect it while maximizing the productivity gains produced by the made improvements. In order to help the developer with the *Reflection* task, we created a *Feedback Collection Guidance* that helps him to decide what information is viable. He creates a feedback entry for the reengineers by updating the *Feedback Entry List*, whenever a task he carries out is characterized by any of the following descriptions: (1) the task deals with fixing a problem, e.g. instructions given are not valid, (2) the task is cognitively simple, e.g. copy and paste is performed, or (3) the task is repetitive. Thereby, each feedback entry contains the following information: (1) *Description*: What needed to be done?, (2) *Frequency/Duration*: How often was it done?, (3) *Location*: Which artifacts were affected?, and (4) *Type*: What type of activity was performed (e.g. create, change, delete, lookup)?.

2.2 Feedback Evaluation during Reengineering

In the *Reengineering Activity*, the reengineer has to evaluate the given feedback and derive actions to adapt the instructions or tool chain and therefore the generated source code accordingly. As depicted in Figure 1, we introduced a specific *Required Actions Evaluation*

task that is performed initially. Not all feedback entries have the potential to make the migration more efficient. Thus, each feedback entry is systematically assessed using the *Feedback Assessment Guidance*. For each feedback entry, a reengineer has to understand and sketch the potential actions in order to address it. In our case study, we identified two possible actions that may result: Instruction adaptation or tool chain adaptation. The instructions as well as the tool chain can either be revised or extended. Revision is necessary in the presence of a flaw, e.g. a faulty transformation, to increase the effectiveness. In contrast, extension may increase the efficiency by increasing the amount of the generated code or providing missing information in the documentation. As a result, an automatic conversion may be realized. The identified potential actions are added to the *Action List*.

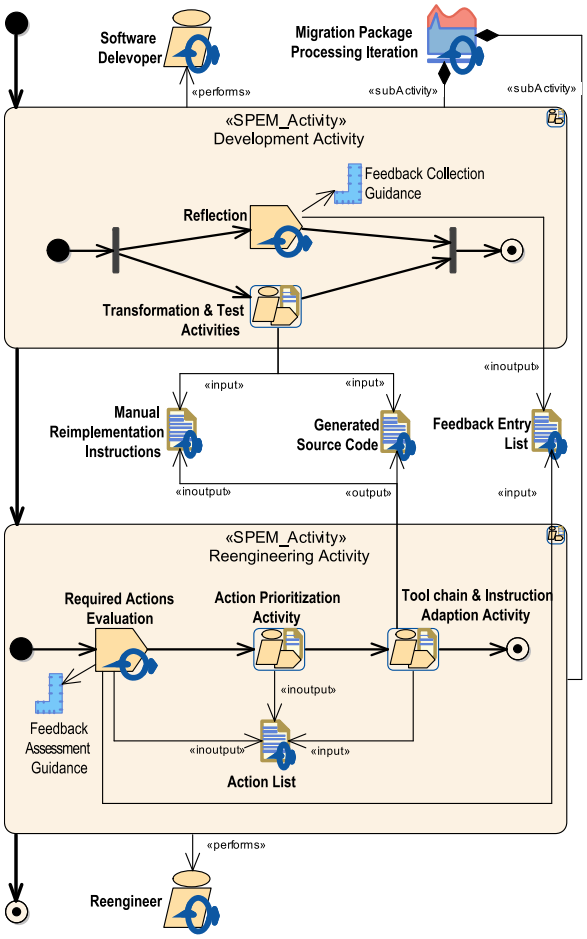

Figure 1: Overview of the Feedback-Enhanced Method

After potential resulting actions have been determined, the reengineer needs to prioritize them within the existing *Action Prioritization Activity*. In order to do this, he has to evaluate the estimated effort in relation to the estimated benefit. As a result, he may also decide to ignore the action and thus the related feedback entry. Otherwise, the prioritized actions based on the feedback entry list are treated in the same way as other project actions, e.g. they are managed in a project-wide issue tracking system.

The prioritized Action List is the input of the *Tool chain & Instruction Adaption Activity*, where the scheduled actions are performed and as a result the migration artifacts are updated.

3 Related Work

As software migration has been an active area of research for quite some time, several methods have been proposed [2]. To the best of our knowledge, no emphasis has been set on how to systematically exchange feedback between the developers and the reengineers in order to increase the effectiveness and efficiency of the overall process. This topic is also underrepresented in experience reports. For example, in [4] and [5], case studies are described which indicate, that some feedback in terms of experiences during the development was used to adapt and extend the tool chain. However, no details are given.

4 Preliminary Results and Future Work

This method was developed in an industrial context. It has been applied in the pilot migration of a legacy application system consisting of about 5 KLOC written in PL/SQL and 2 K declarative elements defined in a 4th generation language (4GL). The application was migrated by a team of two reengineers and two developers. Albeit being a considerable small project, applying the described method already supported the systematic improvement of the overall efficiency and effectiveness. We believe that this method can also be applied in large-scale migration projects. As development activities are often outsourced in these projects, the information gap between the groups of reengineers and developers is much bigger, such that applying our method should be even more beneficial.

5 Literature

[1] Grieger, M.; Güldali, B.; Sauer, S.: Sichern der Zukunftsfähigkeit bei der Migration von Legacy-Systemen durch modellgetriebene Softwareentwicklung. In Softwaretechnik-Trends, vol. 32, no. 2, pp. 37-38, 2012.

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

[3] Object Management Group: Software & Systems Process Engineering Meta-Model Specification, 2008.

[4] Fleurey, F. et al.: Model-driven Engineering for Software Migration in a Large Industrial Context. In Proc. of MODELS 2007, pp. 482–497, 2007.

[5] Winter, A. et al.: SOAMIG Project: Model-Driven Software Migration Towards Service-Oriented Architectures. In Proc. of MDSM 2011, vol. 708 of CEUR Workshop Proceedings, p. 15–16, 2011.