A Study on Tool Support for Refactoring in Database Applications

Hagen Schink[1], Janet Siegmund*[2], Reimar Schröter*[1], Thomas Thüm[3], and Gunter Saake[1]

[1]University of Magdeburg, [2]University of Passau, [3]TU Braunschweig

Abstract

Refactoring is a widespread method to improve the structure of an application's source code without affecting the application's behavior. However, since refactorings are defined for single programming languages or programming paradigms, refactorings do not consider the interaction of source code of different programming languages. Thus, refactoring can break applications written in different programming languages. We found that our tool improves the productivity regardless of the participants' programming experience, but there is also room for improvement regarding support for certain refactoring tasks.

1 Introduction

The term *refactoring* describes changes that alter the structure but not the semantics of source code. Refactoring is a widespread methodology in software development and is supported by state-of-the-art IDEs, such as Eclipse. Developers use refactorings to increase the maintainability of source code and to ease the implementation of new features.

The combination of different general-purpose and domain-specific languages is common in software development. Domain-specific languages allow developers a concise implementation of logic of a certain domain, whereas general-purpose languages support developers in the general application development.

To support refactoring in database applications, we developed the refactoring tool *sql-schema-comparer (SSC)* [3]. The aim of our study was to evaluate whether SSC actually improves the productivity of developers refactoring a database application.

2 The Tool

SSC uses graphs to detect mismatches between the schema of a relational database and the schema expected by the Java source code that accesses the relational database. In Java, the expected schema can be defined with the *Java Persistence API (JPA)*. For instance, for the JPA entity `Department` in Listing 1, SSC creates one graph with a root node *departments* and the two leaves *id* and *name*. Since SSC would create the same graph for a table `departments` with the two columns `id` and `name`, SSC would not detect a mismatch between the expected and the actual schema. We implemented a plug-in that integrates SSC in Eclipse. The plug in highlights affected Java classes, fields, and methods, if it detects a mismatch. For instance, the plug-in marks a Java class annotated with `@Entity` as erroneous that maps a missing table.

Listing 1: JPA entity `Department`

```
1 @Entity
2 @Table(name="departments")
3 public class Department implements Serializable {
4   // fields and setters omitted for brevity
5   @Id
6   public int getId() { return id; }
7   public String getName() { return name; }
8 }
```

3 Study Method

We offered students of a database course at the University of Magdeburg appointments for taking part in our study. The participants had to solve two refactoring tasks on the source code of the open-source projects *Apache Syncope* and *AppFuse* with 77 000 and 24 000 lines of Java code, respectively. Both projects use JPA to access a relational database. For the two refactoring tasks, we prepared two Eclipse projects with database instances on which we applied a *Rename-Column* (Task 1) and a *Move-Column* refactoring (Task 2) beforehand. The refactored databases cause the projects' unit tests to fail. Additionally, for each task we provided the participants with a description of the refactoring applied to the database instances. We assigned participants randomly to a group with or without SSC support. A group with SSC support got an Eclipse environment with the SSC plug-in pre-installed, whereas the group without SSC support got a plain Eclipse for the experiment. The participants had to run the projects' unit test to ensure their changes fixed the broken applications. We measured the development times as metric for productivity for each participant. Before the experiment started, we used a minimal code example to give an introduction to JPA, refactoring, and, if necessary, to SSC. In the introduction, we used the Rename-Column refactoring as example.

*The work of Janet Siegmund and Reimar Schröter has been supported by the DFG grant SI 2045/2-1 and BMBF grant 01IS14017B, respectively.

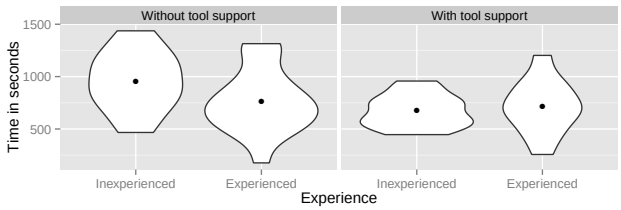

Figure 1: Development time distribution for Task 1.

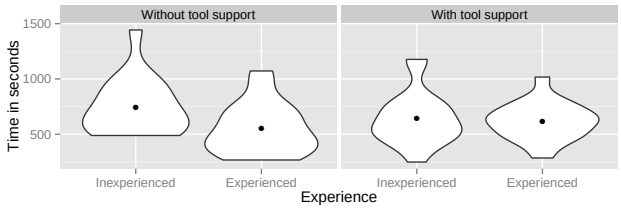

Figure 2: Development time distribution for Task 2.

4 Results

In the experiment, we collected the development times of 76 undergraduate and 3 graduate students. As major confounding parameter, we consider programming experience which we measured based on a questionnaire [2]. Based on the participants' experience, we distinguished *inexperienced* and *experienced* participants in the analysis.

Before we analyzed the data, we cleaned the data set. First, we removed measurements of unsuccessful tasks. We identified an unsuccessful task by a missing successful unit-test run in the unit-test log. Hence, we removed also incomplete tasks, because these miss a successful unit-test run, too. After the removal of unsuccessful tasks, we removed outliers, that is, development times that deviate more than 1.5 standard deviations from the mean. Eventually, we got 67 and 71 results which we considered for the final analysis.

Figures 1 and 2 present the development times grouped by tool support and programming experience. We applied a two-way ANOVA [1] on the results of each task, with the independent variable SSC support and the confounding variable programming experience as the two factors. For Task 1, the groups with SSC support were faster than the groups without SSC support. The significance test shows a significant effect of the variable SSC support ($p < .01$). Considering the means, the performance differs for inexperienced participants by 284.3 seconds and for experienced participants by 53.7 seconds. The results show no significant interaction of SSC support and programming experience.

For Task 2, the groups with SSC support were slower than the experienced group without SSC support, but faster than the inexperienced group without SSC support. The results of the significance test show no significant effect of SSC support, but of programming experience ($p < .05$) on the development time.

5 Discussion and Conclusion

For Task 1, support by SSC allowed participants significant faster adaption of the broken source code to a renamed column in the database schema, regardless of their programming experience. Since all participants attended the introduction, which shares the same rationale with Task 1, we assume that all participants were equally prepared for Task 1. Thus, we conclude that the SSC improved the participants' performance with its ability to mark the Java source code affected by the refactored database schema. A plain Eclipse does not provide such information.

SSC support has no significant influence on Task 2. However, programming experience shows a significant effect on Task 2. The effect of programming experience is independent of SSC support. For Task 2, participants had to adapt the Java source code to a moved column in the database schema. Therefore, the participants had to move specific methods from a source to a target Java class. However, SCC only highlights the affected methods in the source class, but not the target Java class. Thus, participants with SSC support were forced to find a way to complete the adaption without SSC. We assume that experienced users were faster to find the target class, irrespective of SCC support. One possible reason for the better performance of experienced participants could be the ability of experienced users to faster adapt to the unknown code base.

The results of Task 1 suggest that SSC can improve the performance of developers adapting Java source code to a database refactoring. However, as the results of Task 2 suggest, advantages of SSC may be diminished by an incomplete support of different refactoring tasks. However, supporting Task 2 can be achieved by exposing features that exist in SSC, but not in its Eclipse plug-in used in the experiment. Thus, in our future work, we will focus on integrating more features of SCC into its Eclipse plug-in and on the evaluation of different refactoring scenarios.

References

[1] G. Casella. *Statistical Design*. Springer-Verlag New York, 2008.

[2] J. Feigenspan, C. Kästner, J. Liebig, S. Apel, S. Hanenberg, and K. Christian. Measuring Programming Experience. *IEEE International Conference on Program Comprehension*, pages 73–82, 2012.

[3] H. Schink. Sql-Schema-Comparer: Support of Multi-Language Refactoring with Relational Databases. *International Working Conference on Source Code Analysis and Manipulation*, pages 164–169, 2013.