A Change Impact Analysis Case Study: Replacing the Input Data Model of SoMoX*

Benjamin Klatt, Martin Küster, Klaus Krogmann, Oliver Burkhardt
FZI Research Center for Information Technology
Haid-und-Neu-Str. 10-14, 76131 Karlsruhe, Germany
{klatt,kuester,krogmann,burkhardt}@fzi.de

1 Introduction

Change impact analysis aims to provide insights about efforts and effects of a change to be expected, and to prevent missed adaptations. However, the benefit of applying an analysis in a given scenario is not clear. Only a few studies about change impact analysis approaches compare the actual effort spent implementing the change with the prediction of the analysis [4]. To gain more insight about change impact analysis benefits, we have performed a case study on changing a software's input data model. We have applied two analyses — using the Java compiler and a dependency graph based approach — before implementing the actual change. In this paper, we present the results, showing that i) syntactically required changes have been predicted adequately, iii) changes required for semantical correctness required the major effort but were not predicted at all, and iii) tool support for change impact analysis still needs to be improved.

2 Change Scenario Under Study

The case study was performed on the open source software SoMoX[1]. SoMoX is a software analysis tool capable to re-engineer a component structure from a software's implementation. It consists of a set of Eclipse plug-ins with approximate 9,000 lines of Java code. SoMoX consumes an Abstract Syntax Tree (AST) model as input data model to detect components based on hierarchical clustering of basic components detected from classes and interfaces.

As shown in Figure 1, subject of the change is the above mentioned AST model. The original model, provided by the SISSy tooling[2], should be replaced with the one provided by the MoDisco tooling [1].

Both data models are based on the Eclipse Modeling Framework (EMF) and are designed to describe the software entities of object-oriented programs. They have been modeled as Ecore meta models, their Java implementations have been generated by EMF and are provided as Eclipse plug-ins.

Due to the structural similarities of the two models, we expected only minor structural differences and

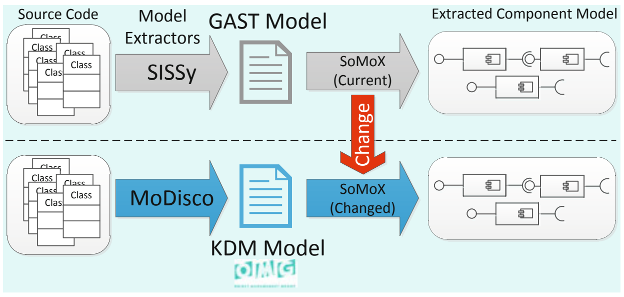

Figure 1: SoMoX Change Scenario

obvious namespace changes. As described in the following section, a change impact analysis was intended to provide insight into the detailed efforts and to help not to miss any locations that need to be adapted.

3 Change Analyses Performed

The change scenario under study requires an approach working on the existing implementation of a software. It must be capable of analyzing based on an input with granularity of types (i.e. classes and interfaces) or more detailed level (e.g. methods). The analysis result must return expressions, representing code locations that need to be changed. Furthermore, the approach should determine the change impact in an automated, non-iterative manner, to get an estimation in advance of the implementation.

Evaluating existing approaches surveyed by Lehnert [4], none of them provides a public available tooling to be applied in the case study. The most promising candidate, JRipples [2], had to be discarded because of its iterative workflow of analyzing and implementing a change. In addition, it could not be applied due to a technical limitation on systems consisting of a single Eclipse project only.

Finally, two approaches have been selected to be applied: First, facilitating compiler messages as a naive, straight-forward, and low-overhead approach. Second, executing a dependency graph query as a more mature and widely known approach.

3.1 Compiler-Messages

The Eclipse Java Development Tools (JDT) prepare and present messages of the Java compiler. Removing the current input data model of SoMoX from the workspace leads to such messages presenting according compiler errors. The JDT provide statistics about

*This work was supported by the German Federal Ministry of Education & Research (BMBF), grant No. 01IS12012B.

[1]http://www.somox.org

[2]http://www.sqools.org/sissy/

Category	Compiler	Query
Estimated	700	710
Estimated And Changed	513	710
Estimated Not Changed	187	0
Syntax only:		
Changes Not Estimated	200	3
Implemented Changes	713	713
Precision	73.29%	100.00%
Recall	71.95%	99.58%
Including Semantic:		
Changes Not Estimated	752	555
Implemented Changes	1265	1265
Precision	73.29%	100.00%
Recall	40.55%	56.12%

Table 1: Change Impact Analysis Results

the compiler messages and links to code locations estimated as impacted by the missing data model.

3.2 Dependency Graph Analysis

For the case study, a dependency graph query according to Lee et al.[3] has been implemented based on the MoDisco infrastructure. The query finds all *TypeAccess*, *AbstractMethodInvocation*, *SingleVariableAccess*, and *ImportDeclaration* references in the packages under study (i.e. starting with ”org.somox”) referencing to elements in the input data model packages (i.e. ”de.fzi.gast”).

Using a MoDisco discoverer extracting Java AST and KDM inventory models, the query result view provides statistics as well as links to the according code locations in a Java editor.

Implementing the query took approximate one day and was done by a developer already familiar with the MoDisco infrastructure and the Java AST meta model the query was written for.

4 Evaluation of the Results

To evaluate the benefit of the analysis, the two analyses have been performed before the actual change was implemented. During the implementation, the modified locations have been tracked in terms of changed locations. If more than one location was changed within a single line of code, a change has been counted for each location. For example, replacing ClassA with ClassB for the var1's type and the constructor call in the statement ClassA var1 = new ClassA(); is counted as two changes.

Table 1 presents the results of the two analyses. The first section of the table shows, that the query approach estimated more required changes (Hits) and less false positives (Miss). For example, the compiler does not identify member accesses (e.g. method calls) on unresolved data types. Furthermore, it reports false positives such as classes containing impacts but do not require any changes themselves. We distinguish syntactical changes, required to successfully compile SoMoX, and semantical changes, required to produce the same detection results as before.

The semantic changes are primarily caused by manual adaptations to the generated SISSy AST model. They are used to pre-calculate insights about software elements. While this is not possible for the new data input model, 42 helper methods containing 552 lines of code were added to the SoMoX implementation.

The syntactical changes, not estimated by both approaches, are string comparisons of file extensions.

In summary, the query approach provides better results compared to the compiler approach. For syntactical impacts, it achieves 100% over 73.29% precision and 99.58% recall over 71.95% recall which are both good results. However, taking semantical changes into account, the recall is significantly reduced for both approaches to 56.12% respectively 40.55% percent.

5 Conclusion and Outlook

In this paper, we presented the results of our case study on an impact analysis for changing the input data model of SoMoX. We identified no public available implementation of an applicable change impact analysis. As surveyed by Lehnert [4], many approaches are available (∼160 in the survey). But only a few describe evaluation experiments (∼40). Even those rarely compare their prediction with actual change implementations. For example, Toth et al. [6] compared their tool's estimation with the estimation of programmers. Also, Lindvall et al. [5] have compared programmers' estimations with the actual implemented changes. We further described the compiler-based and dependency graph-based approaches that we have chosen, implemented and applied. The results show that the query-based approach adequately estimated the required syntactical changes, while both were not able to estimate the semantic changes which required the major amount of effort.

We published our MoDisco-based change impact query on the SoMoX website. In addition, we currently wrap this query as an Eclipse plug-in for simplified reuse by others.

References

[1] H. Bruneliere, J. Cabot, F. Jouault, and F. Madiot. MoDisco: a generic and extensible framework for model driven reverse engineering. In *ASE'10*.

[2] J. Buckner, J. Buchta, M. Petrenko, and V. Rajlich. JRipples: A Tool for Program Comprehension during Incremental Change. In *IWPC'05*.

[3] M. Lee, J. Offutt, and R. T. Alexander. Algorithmic Analysis of the Impacts of Changes to Object-Oriented Software. In *TOOLS'00*.

[4] S. Lehnert. A Review of Software Change Impact Analysis. Technical report, Technische Universität Ilmenau, 2011.

[5] M. Lindvall. Evaluating Impact Analysis - A Case Study. *Empirical Software Engineering*, 2(2), 1997.

[6] G. Tóth, H. Péter, A. Beszédes, T. Gyimóthy, and J. Jász. Comparison of Different Impact Analysis Methods and Programmer's Opinion - an Empirical Study. In *PPPJ'10*.