A Methodology for Impact Analysis Based on Model Differencing

Klaus Müller, Bernhard Rumpe
Software Engineering
RWTH Aachen University
mueller@se-rwth.de, rumpe@se-rwth.de

1 Introduction

A software system typically has to be changed frequently to adapt the system to new or changing requirements or due to bug fixes. One crucial problem is that every kind of change can introduce severe errors into the software system and that it is difficult to predict in what way which parts of a software system are potentially affected by a change. Impact analysis approaches cope with this problem by trying to identify the potential consequences of changes [1].

In model-based software development, models are usually transformed into concrete implementations [2]. Even though code generators can automatically generate essential parts of a software system, it is usually still required to create and maintain further handwritten artifacts such as source code. These artifacts have to be integrated into the generated parts of the software system and, thus, they sometimes heavily depend on the generated artifacts. For example, a code generator might generate the database schema based on a UML class diagram. If developers introduce a handwritten source code file which contains SQL queries that access this database, the source code file depends on the generated database schema and consequently also on the UML class diagram. Hence, model changes can have tremendous impact on the handwritten artifacts.

In this extended abstract, we discuss a methodology for developing and applying a model-based impact analysis approach, in which explicit impact rules can be specified in a domain specific language (DSL). These impact rules embody what kind of model changes have what kind of impact. Based on such impact rule specifications, impact rule implementations are generated, which check the specified conditions and output the defined impact.

The main advantage of defining explicit impact rules is that they allow formalizing knowledge about known dependencies and characteristics of a software system. As the impact rules describe the impact of model changes, it is possible to create a checklist that informs developers about the impacts of all model changes that have been detected in a model differencing step. The resulting checklist, thus, contains concrete hints about the development steps that are (potentially) necessary to adapt the system to the model changes. Due to this, the checklists can simplify the evolution process, as developers can work through the checklists. The motivation for creating such checklists is that developers might forget to perform certain development steps that are necessary after specific model changes. This particularly holds in a complex software system.

In the next section, we elaborate on the methodology for developing and appyling the approach. Results of a case study dealing with the impacts of UML class diagram changes can be found in [3].

2 Methodology to Generate Checklists

Our methodology proposes an impact analysis approach that is composed of two steps: the identification of model differences and the application of impact rules on these differences. As a result, the approach produces a checklist which can be ticked off.

Subsequently, Subsection 2.1 outlines the steps that have to be performed to set up this impact analysis approach. After that, Subsection 2.2 outlines the steps that have to be carried out to apply the approach.

2.1 Setting up the Impact Analysis Approach

At first, a model differencing tool has to be chosen to be able to perform model differencing. If the chosen model differencing tool expects input models of a certain type, but the original input models have another type, a model converter has to be implemented or an existing one has to be integrated into the tool chain to allow for differencing the input models.

One problem that has to be considered in the context of model differencing is that a completely automatic approach to model differencing cannot infer the differences correctly in all cases [4]. Because of this, we propose to allow users to integrate knowledge of how specific model elements changed in so-called user presettings. Hence, user presettings have to be derived that fit to the corresponding input model type. Furthermore, the model differencing tool needs to be extended to be able to process user presettings [4].

These two steps are the only required steps to be able to calculate the model differences. Next, the steps that are necessary to set up the impact analysis part of the approach are sketched.

The impact analysis approach that results from applying the proposed methodology relies on impact rules capturing the consequences of changes in particular types of models. In an impact rule the user is free to define what kind of change leads to what kind of impact. To improve the comprehensibility of an impact rule, the methodology proposes a simple DSL, in which it can be specified which conditions have to be fulfilled by a model difference so that a certain checklist hint is created.

```
1  impactRule "IRExample" {
2    description = "Example description"
3    severity = critical
4    relevantFor = "mueller@se-rwth.de"
5
6    impact {
7      renamedClass() =>
8        "Implement data migration."
9    }
10 }
```

Figure 1: Simple impact rule example

A very simple example of an impact rule written in the DSL is illustrated in Figure 1. At first, a description indicating what the impact rule is used for (line 2) is denoted, then it is defined how critical violations against the impact rule are (line 3) and for which persons the hints are relevant for (line 4).

In the subsequent part, it is defined which conditions have to be fulfilled by a model difference to result in the creation of the subsequently given checklist hint (line 7 – 8). According to Listing 1, a checklist would inform the developer about the necessity to implement a data migration if a class was renamed. An impact rule can contain zero or multiple blocks of such condition parts and according checklist hints. Moreover, the condition part can consist of multiple conditions that can be combined using the logical operators && and || known from Java. For each type of model change which can be found in the model differencing step and which is relevant for the impact analysis, we propose to derive a condition which checks whether the particular type of model change occurred. For instance for UML class diagrams, there would be conditions such as `renamedClass` (see line 7 of Listing 1) or `addedAssociation` [3]. These different conditions need to be implemented in the checklist tool so that the conditions can be referenced in the condition part of the impact rule DSL.

As soon as a first set of conditions has been implemented, concrete impact rules can be defined using the impact rule DSL. An impact rule generator will generate implementations of the impact rules out of the impact rule specifications written in the DSL. In some situations it can be necessary to extend this generated implementation and to add handwritten parts to a handwritten subclass [3].

2.2 Applying the Impact Analysis Approach

After having executed the steps listed in the previous subsection, the tool chain contains the required parts to identify the impacts of model changes. The workflow to apply this impact analysis approach to generate checklists is outlined in the following.

If not all impact rules should be executed when creating the checklist, the checklist tool first needs to be configured by defining which impact rules should (not) be invoked in the checklist generation. By default, all impact rules are taken into account.

Afterwards, the developers have to decide for which pairs of input models the checklists should be generated. If a model converter had been integrated into the tool chain, the next step is the invocation of the model converter to produce models that can be processed by the model differencing tool. Finally, the model differencing tool is invoked for the potentially converted pairs of input models.

Next, the checklist generator is called for the resulting difference model. Every difference contained in the difference model is passed to the impact rules. Each impact rule then analyzes the current model difference and creates a list of hints at further (potential) development steps, if the difference is regarded as relevant. These different hints are finally merged into a checklist, together with further information such as a list of detected model differences.

Before performing the development steps that are contained in the resulting checklist, developers need to verify that the reported model differences are correct. If wrong model differences have been reported, developers have to provide user presettings to fix these problems. In this case, the model differencing tool has to be invoked again and the subsequent steps have to be repeated.

Remarks This extended abstract discusses a generalization of previous work [3].

References

[1] S. A. Bohner. *A graph traceability approach for software change impact analysis.* PhD thesis, George Mason University, Fairfax, VA, USA, 1995.

[2] R. France, B. Rumpe. Model-Driven Development of Complex Software: A Research Roadmap. In *Proc. Future of Software Engineering (FUSE'07).* Pp. 37–54. 2007.

[3] K. Müller and B. Rumpe, "A Model-Based Approach to Impact Analysis Using Model Differencing," *in Proc. International Workshop on Software Quality and Maintainability (SQM'14), ECEASST Journal*, vol. 65, 2014.

[4] K. Müller and B. Rumpe, "User-Driven Adaptation of Model Differencing Results," *International Workshop on Comparison and Versioning of Software Models (CVSM'14), GI Softwaretechnik-Trends*, vol. 34, no. 2, pp. 25–29, May 2014.