Reconstructing Development Artifacts for Change Impact Analysis*

Kiana Rostami[1], Michael Langhammer[1], Axel Busch[1]
Joshua Gleitze[2], Robert Heinrich[1], Ralf Reussner[1]
Karlsruhe Institute of Technology (KIT),
[1]{rostami,langhammer,busch,heinrich,reussner}@kit.edu, [2]joshua.gleitze@student.kit.edu

Abstract

Software architectural models are widely used to represent the structure of software systems. Software systems need to evolve continuously during their life time, for instance, to adapt to new requirements. During the evolution various change requests have to be implemented. However, analysing the architecture of a system alone does not provide sufficient information for an adequate estimation of the impact resulting by such change requests. In addition, many other development artifacts, such as test cases, have to be considered. Creating models of these artifacts by hand is time-consuming and error-prone. In this paper, we present an approach that automatically extracts development artifacts and annotates them to a software architectural model.

1 Introduction

Modern component-based software development is based on an explicit software architectural model. Such a model enables analysis on quality attributes of the software system, like performance or maintainability. As maintenance takes the lion's share in software's life cycle, it would be desirable to additionally have a model allowing quality analysis for software systems in change scenarios. Modeling a typical software architecture, that comes with many classes and implicit design decisions, by hand can be a very time-consuming and tedious process. Several approaches exist to automate the re-engineering process, automatically extracting components and their relationships from existing software. Based on an architectural model, maintainability analysis usually results in a task list containing several tasks needed to implement a change request [4]. For the maintainability analysis, additional management and technical artifacts (e.g. test cases) need to be considered, as they can also be affected by a change request. However, most existing re-engineering tools are limited to analyse source code [1].

This paper presents the approach Source Code Model eXtractor for Karlsruhe Architectural Maintainability Prediction (SoMoX4KAMP) to reconstruct further management and technical artifacts during the re-engineering process, omitting the effort of creating the artifacts by hand. We extend the reverse engineering approach approach Source Code Model eXtractor (SoMoX) [2] to consider test cases, configuration files, and other technical descriptions that are essential artifacts in modern software projects. The maintainability approach Karlsruhe Architectural Maintainability Prediction (KAMP) [4, 5] uses these artifacts to derive more accurate task lists needed to implement change requests.

2 Foundation

Our work is based on the Palladio Component Model (PCM), KAMP, and SoMoX.

The PCM [3] is a component-based architectural metamodel for software systems. It roughly consists of components, interfaces, signatures, and the "provides" and "requires" relations between components and interfaces. Based on the architectural model, the PCM can be used to conduct model-based analysis, such as performance prediction.

If the PCM should be applied to existing projects that do not have an explicit or no up-to-date architectural model, users face the problem of creating it. To avoid the effort of building PCM model instances from existing code manually, it is possible to use the reverse engineering approach SoMoX [2]. It uses a combination of different metrics to reverse engineer PCM components and PCM interfaces from Java code. One metric, for instance, is the package metric, which determines if classes in one package should be considered as one component.

KAMP [4, 5] is a scenario-based maintainability approach that uses a formal software architectural metamodel (i.e. PCM) to estimate change propagation. Using a further metamodel for annotating the context information, KAMP can also consider the technical and organizational tasks caused by a change request. Fig. 1 shows an excerpt of FieldOfActivityAnnotation metamodel used for the context information.

3 Approach

To accurately predict change propagation in a software system, information about many artifacts is necessary.

*This work was partially supported by the DFG (German Research Foundation) under the Priority Programme SPP1593.

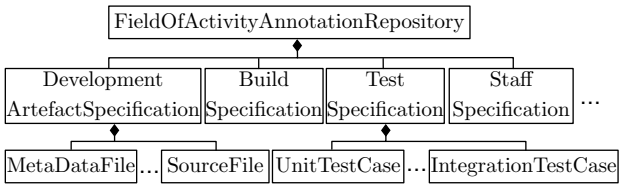

Figure 1: FieldOfActivityAnnotation (excerpt) [5]

Besides source code, test cases are for example typical artifacts needed to be considered by software architects. To include them in the prediction process, they need to be modeled explicitly in an own model and linked with the originate architectural model. To omit the necessity of modeling the additional artifacts manually, we created SoMoX4KAMP, which is able to reverse engineer them automatically.

SoMoX4KAMP is executed together with SoMoX and uses the results of SoMoX. It additionally uses information from source code files as well as further files and meta data information. The output of SoMoX4KAMP is an instance of KAMP's *FieldOfActivityAnnotation* metamodel (as illustrated in Fig. 2). We currently implemented the reverse engineering for build specifications, test specifications, and staff members, which we will elaborate in the following.

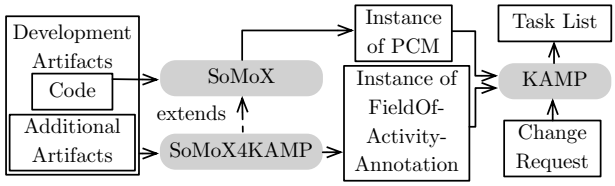

Figure 2: Overview of the holistic approach

3.1 Build Specification

Build specifications define how a specific software system can be build, for instance, using an automatic build system. They are usually stored in a technology specific representation. By convention, a build specification file describes build rules only for files that are in or in a subfolder of the folder the build specification file is in. SoMoX4KAMP detects build specifications by their name (e.g. `pom.xml`, `*.gradle`, `Makefile`, etc.) and uses this convention as a heuristic to find all files affected by them. It then adds each specification as a *BuildSpecification* to the resulting *FieldOfActivityAnnotation* model and assigns it to all components having at least one file affected by it.

3.2 Test Specification

Test specifications are artifacts used to test software. In our approach, we only looked at tests written in Java. This includes unit, integration and system tests, but also helper classes. To find test specifications, SoMoX4KAMP first needs to detect source code files that contain test cases. To do so, it iterates over all source files and consults so called *TestDetectors*, which analyse whether a given source code file contains a test, using different heuristics. For instance, a file that is part of a component is never a test specification. A file referencing the JUnit test framework or placed in a test folder according to Maven's Standard Directory layout is a test specification.

Once detected, SoMoX4KAMP statically analyses each test specification source file to search for the components tested by it. It iterates over all method calls made in the tests and follows them transitively until they reach a component. The specification is then added as a *TestSpecification* to the resulting *FieldOfActivityAnnotation* model and assigned to the components found by the search.

3.3 Staff Members

For implementing future change requests it is important to document the developers of a component [5]. Staff members are currently added to the resulting *FieldOfActivityAnnotation* model instance by analysing comments in Java source code and looking at information from version control systems (VCS). SoMoX4KAMP finds Javadoc's `@author` tags in all comments in Java source files. If no `@author` tags can be found in a file but the file is checked into a VCS, editing information from the VCS (like `git`'s `blame`) is used to determine the file's editors. The authors or editors found for a file are then added as *Developers* of the reverse engineered component the file is a part of.

4 Conclusions

We presented SoMoX4KAMP, extending an architectural model reconstruction approach to automatically extract development artifacts (e.g. test cases) for a maintainability analysis approach. Using other development artifacts than code can, by this, improve the estimation of the effort caused by a change request.

References

[1] S. Ducasse and D. Pollet. "Software architecture reconstruction: A process-oriented taxonomy". In: *Software Engineering, IEEE Transactions on* 35.4 (2009), pp. 573–591.

[2] K. Krogmann. "Reconstruction of Software Component Architectures and Behaviour Models using Static and Dynamic Analysis". PhD thesis. KIT, 2010.

[3] R. H. Reussner et al. *Modeling and Simulating Software Architectures – The Palladio Approach*. MIT Press, 2016. 408 pp.

[4] K. Rostami et al. "Architecture-based Assessment and Planning of Change Requests". In: *QoSA*. ACM, 2015, pp. 21–30.

[5] J. Stammel. "Architekturbasierte Bewertung und Planung von Änderungsanfragen". PhD thesis. KIT, 2015.