Model-Driven Reverse Engineering of Technology-Induced Architecture for Quality Prediction

Yves R. Kirschner

Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany

Abstract

In software engineering, techniques of continuous integration allow short test cycles and thus fast feedback cycles. However, if there is a need to change the deployment or update the functionality, it becomes difficult to assess the impact of such changes on performance. In this context, we present in this paper an extensible approach for reverse engineering of component-based systems for quality prediction. We build on the principles and techniques of model-driven engineering to analyze a system at the model level and to enable subsequent optimization.

Keywords

Analytical models, Object oriented modeling, Performance evaluation, Predictive models, Reverse engineering, Software architecture

1. Introduction

Component-based software development is characterized by the concept of developing systems by integrating reusable components that interact with each other and form a clearly defined software architecture. However, the resulting advantages, such as improved maintainability and scalability, also face new challenges. The increased number of software components brings to the fore the importance of compliance with standards for rapid development and sustainable maintenance. To perform this task, a new developer must first understand the architecture of the component-based systems. One way to improve this understanding is to automatically restore and create an architecture model with the system properties relevant for developers. Model-driven reverse engineering techniques can be used for this purpose. It is the process of understanding software and creating a model suitable for documentation, maintenance or reengineering.

Garcia et al. conclude in [1], that clustering of software entities is the almost uniformly applied method for automated architecture recovery. In most cases, a graph structure is generated based on dependencies in the source code, so that components can be reconstructed using clustering or pattern matching. However, our newly proposed approach aims at using mapping rules on a structural level for the reconstruction of architectural models from source code. We assume that components with their interfaces and distribution can often be explicitly determined by the

ECSA'21: Doctoral Symposium, September 13–17, 2021, Växjö, Sweden

yves.kirschner@kit.edu (Y. R. Kirschner)

technology used, such as application frameworks like the Spring framework or API specification like JAX-RS. By considering technologies, we expect better results in reverse engineering, since heuristics such as cohesion or coupling do not have to be used to determine components from source code.

The goal of our proposed approach is to support the creation and maintenance of software architecture models for quality prediction. For this purpose, we want to capitalize on reusable descriptions of concepts for the reverse engineering of models for quality prediction. To achieve such a goal, the following research questions are formulated:
RQ1 How do the technologies used induce the software architecture?
RQ2 How can knowledge about technologies be implemented as rules for reverse engineering?
RQ3 How can these rules be composable for different technologies for a software system?

2. Foundations

The application of model-driven software development techniques to solve reverse engineering problems is called Model-Driven Reverse Engineering (MDRE). [2] defines MDRE as the creation of descriptive models from existing systems that have been previously created in some way. MDRE is about transforming heterogeneous software development artifacts into homogeneous models. According to [3], the following two steps form an MDRE process: 1. conversion of the software system to be analyzed into a set of models of software development artifacts without losing the necessary information; 2. using these models to generate the desired output models through model transformations.

The motivation of Model-Driven Quality Prediction (MDQP) is to enable the early design time prediction of software systems to determine quality characteristics such as performance, reliability or data protection. MDQP attempts to automate the process of analyzing the information available in the model and provide software developers and architects with these analysis values as early as possible as a basis for design decisions.

An example of a MDQP approach would be the so-called Palladio approach [4]. With the Palladio Component Model (PCM), this approach provides a modeling language for the definition of component-based software architectures for quality prediction. The model defines interfaces that describe a set of services. A component can either require or provide interfaces and describes the resource demand for each provided service. Furthermore, this approach provides an integrated modeling environment, the so-called Palladio-Bench, based on the Eclipse IDE. By solving a PCM instance analytically or simulation-based, the Palladio approach enables quality predictions. We use the PCM as the underlying architectural model because it allows model-driven quality prediction. However, the presented underlying idea of the approach is not limited to the PCM. For example, for the pure reverse engineering of the software architecture, a UML-based model could also be applied as a resulting model.

3. Proposed Approach

The idea of the approach we propose is to model the knowledge of the domain of the technologies used in component-based software development in order to reverse engineer the architecture from artifacts. This knowledge could describe how a component is implemented using a certain framework. Our approach is module-based and comes with a predefined set of modules for model discovery, understanding and generation. This allows our approach to be easily extended by project-specific rules or implementations. Our approach is to integrate the following steps into the continuous integration of a software system: 1. Discovery: Parsing the existing artifacts, accumulating structure and behavior information about these artifacts in EMF-based models. 2. Understanding: Analyzing these models using rules represented by model-to-model transformations. Detected concepts are stored in the decorated model. 3. Generation: Generate behavioral models based on the source code model and the decorated model.

3.1. Model Discovery

The first step of our reverse engineering approach is to obtain a model from existing artifacts that allows a uniform view of the software system. Thereby artifacts that are written during the development of a software system, e. g. source code or other configuration files like deployment descriptors, are taken into account. In order for these models to provide a uniform view, they must conform to a given metamodel that expresses the selected artifacts. This metamodel could be programming language specific or map other configuration files. Moreover, these metamodels are created as Ecore metamodels in order to enable uniform modeling. The actual structure of these models is realized in model-driven reverse engineering by so-called discoverers, which depend on the associated metamodel. In our approach, discoverers are provided by additional modules and can support different types of artifacts.

3.2. Model Understanding

The second step is the main step of our reverse engineering approach. In this step, the previously generated models are used to effectively achieve the desired reverse engineering scenario. During this step, these models are analyzed by using so-called rules. The underlying idea of the rules is to capture domain knowledge about technologies with them. We use this knowledge to reverse engineer the architecture of a system in which this technology is used. For this purpose, rules capture how a certain concept is implemented in a technology and, which effects this concept has on the architecture of the system. These rules are expressed as model-to-model transformations.

3.3. Model Generation

In the third step the behavior of the methods of the interfaces of the recognized components is transformed into a PCM instance. In this step, parts of the Palladio-Bench are used to transform the behavior given in the Java model into an abstract behavior on component level. For this purpose, not only the Java model but also the references to the previously recognized concepts are entered into the Palladio-Bench. Links to the original source classes are stored here for each

architecture model entity. By this mechanism, any later analysis result on architecture level can be traced back to low-level classes and methods.

4. Expected Results and Evaluation

The expected main contribution is the design and development of the model-driven reverse engineering approach described in the previous section. Where the focus will be on the rules for a diverse set of technologies. We plan to evaluate both the framework and these rules. Evaluation methods will include industrial case studies as well as reference applications for reverse engineering. In this evaluation, we will define goals, questions whose answers will indicate whether the goal has been achieved, and metrics that represent measurable answers to the questions. On this basis, we will compare the quality predictions of our own reverse engineering results with related approaches.

5. Related Work

Tzerpos and Holt present ACCD [5], an algorithm for communication-driven clustering that groups files based on name patterns. Mancoridis et al. present Bunch, a tool for generated decompositions using similarity measurements, which uses upscaling algorithms to group files into clusters based on coupling and cohesion [6]. Andritsos et al. present LIMBO [7], an algorithm for information-theoretical clustering for software that uses hierarchical clustering based on similarities between groups of files to reverse engineer an architecture. Even though each of these reverse engineering methods has a different principle, all of these methods divide source code entities into mutually exclusive clusters, each based on a dominant principle such as cohesion and coupling or naming patterns. Müller et al. present JQAssistant, an approach to create a uniform data source for software analysis and visualization [8]. JQAssistant offers the possibility to extract a model of the software architecture by graph querying. However, only dependencies on the level of methods are considered, so that there is no possibility to process fine granular information, which is needed for the creation of models for quality prediction.

Garzón et al. propose an approach to reverse engineer object-oriented code into a unified language for both object-oriented programming and modeling [9]. By using an incremental and rule-based approach, UML class diagrams and state machines can be mixed with the associated source code. However, these rules cover only the fundamental object-oriented constructs and no special technologies nor are these UML diagrams suitable as a basis for quality prediction. Klint et al. create with RASCAL a domain-specific language, which integrates source code analysis, transformation and generation on the language level [10]. The application purpose of RASCAL is code analysis and manipulation, e. g. for refactoring. However, this code transformation does not support the generation of models which have a different meta-model than the code.

Raibulet et al. compare in depth fifteen different model-driven reverse engineering approaches in their literature review and find that both these approaches and their areas of application are versatile [3]. In this respect, MoDisco is the most related approach in a comprehensive scope. Bruneliere et al. developed MoDisco [11], a generic and extensible model-driven reverse engineering approach. It provides support for Java, JEE and XML technologies to generate

model-based views of the architecture. Although it is extensible, it does not support direct reuse of common concepts and combination of several technologies is not supported.

6. Conclusion

In this paper, we created an overview of our model-driven approach to improve the reverse engineering of component architectures. The main idea is the reverse engineering of components with their interfaces and their distribution from existing software development artifacts such as source code or configuration files under consideration of the used technologies. We expect improved reverse engineering results, since heuristics such as cohesion or coupling no longer need to be used to determine components from the source code. As we want to use knowledge about the technologies used to detect components, we expect our proposed approach to generate models more in line with real architecture. Also we expect technology-specific rules to provide a better understanding of the relationships between a technology and its underlying concept and software architecture. These technology-specific rules allow reuse in projects that use a specific technology.

References

[1] J. Garcia, I. Ivkovic, N. Medvidovic, A comparative analysis of software architecture recovery techniques, in: ASE'13, 2013.

[2] J.-M. Favre, Foundations of model (driven) (reverse) engineering, in: Language Engineering for Model-Driven Software Development, 2005.

[3] C. Raibulet, F. A. Fontana, M. Zanoni, Model-driven reverse engineering approaches: A systematic literature review, IEEE Access 5 (2017) 14516–14542.

[4] R. H. Reussner, S. Becker, J. Happe, R. Heinrich, A. Koziolek, H. Koziolek, M. Kramer, Modeling and simulating software architectures: The Palladio approach, 2016.

[5] V. Tzerpos, R. C. Holt, Accd: an algorithm for comprehension-driven clustering, in: WCRE'00, IEEE, 2000, pp. 258–267.

[6] S. Mancoridis, B. Mitchell, Y. Chen, E. Gansner, Bunch: a clustering tool for the recovery and maintenance of software system structures, in: IEEE ICSM, 1999, pp. 50–59.

[7] P. Andritsos, P. Tsaparas, R. J. Miller, K. C. Sevcik, Limbo: Scalable clustering of categorical data, in: EDBT'04, Springer, 2004, pp. 123–146.

[8] R. Müller, D. Mahler, M. Hunger, J. Nerche, M. Harrer, Towards an open source stack to create a unified data source for software analysis and visualization, in: VISSOFT'18, IEEE, 2018, pp. 107–111.

[9] M. A. Garzón, T. C. Lethbridge, H. I. Aljamaan, O. Badreddin, Reverse engineering of object-oriented code into umple using an incremental and rule-based approach., in: CASCON'14, 2014, pp. 91–105.

[10] P. Klint, T. van der Storm, J. Vinju, Rascal: A domain specific language for source code analysis and manipulation, in: ICSME'09, 2009, pp. 168–177.

[11] H. Bruneliere, J. Cabot, F. Jouault, F. Madiot, Modisco: A generic and extensible framework for model driven reverse engineering, in: ASE'11, ASE'10, ACM, 2011, pp. 173–174.