Architecture-based Analysis of Changes in Information System Evolution*

Robert Heinrich[1], Kiana Rostami[1], Johannes Stammel[2], Thomas Knapp[1], Ralf Reussner[1]
[1]Karlsruhe Institute of Technology (KIT), {heinrich,rostami,reussner}@kit.edu
[2]andrena objects ag, johannes.stammel@andrena.de

Abstract

Software is subject to continuous change. Software quality is determined by large extent through architecture which reflects important decisions, e.g. on structure and technology. For sound decision making during evolution change impacts on various system artifacts must be understood. In this paper, we introduce a new evolution scenario (replacing the database) to an established demonstrator for information system evolution. We demonstrate the application of an architecture-based approach for change impact analysis to identify artifacts affected by the scenario.

1 Introduction

Software-intensive systems, such as information systems, are frequently operated over decades. In industrial practice these systems face diverse changes, e.g. due to emerging requirements, bug fixes, or adaptations in their environment, such as legal constraint or technology stack updates [6]. As a result, the systems change continually which is understood as software evolution [5]. The software architecture is one of the central artifacts of software-intensive systems and is crucial in evolution. Software development and operation involve a variety of organizational and technical roles covering different responsibilities and knowledge. Thus, coordinating and implementing changes is difficult. Although these roles are very heterogeneous, they all use artifacts which are tightly related to software architecture. Reflecting changes in software architecture models helps to identify maintenance tasks for associated artifacts like source code or test cases.

In this paper, we introduce a new evolution scenario "replacing the database" to the *Common Component Modeling Example* (CoCoME) [7] which serves as a common case study on information system evolution. An overview of CoCoME is given in Sec. 2. We use the tool-supported approach *Karlsruhe Architectural Maintainability Prediction* (KAMP) [3] for change impact analysis based on change requests in an architecture model. Sec. 3 introduces KAMP. In Sec. 4, we demonstrate how to apply KAMP to the evolution scenario for identifying artifacts affected by the change request. The paper concludes in Sec. 5.

*This work was partially supported by the DFG (German Research Foundation) under the Priority Programme SPP1593: Design For Future – Managed Software Evolution.

2 CoCoME – A Case Study on Information System Evolution

CoCoME represents a trading system as it can be observed in a supermarket chain handling sales. This includes processing sales at a single store as well as enterprise-wide administrative tasks like ordering products or inventory management. The CoCoME system is organized as a three-layer software architecture which allows for distributing the system on server nodes and for remote communication. Detailed description is given in [7]. CoCoME has been set up as a common demonstrator in a Dagstuhl research seminar. Since CoCoME has been applied and evolved successfully in various DFG and EU research projects, several variants of CoCoME exist, spanning different platforms and technologies, such as plain Java code, service-oriented or hybrid Cloud-based architectures. Various artifacts from development and operation are available, such as requirements specification, design decisions, source code, architecture models, or monitoring and simulation data, that evolved over time.

The new evolution scenario "replacing the database" refers to the plain Java variant of CoCoME. In the scenario, CoCoME faces performance issues. In order to avoid them the company which operates CoCoME decides to replace the existing database. They shift away from a relational database (e.g., MySQL) to a non-relational database (e.g., CouchDB).

3 Architecture-based Change Impact Analysis

KAMP [3] explicitly includes formal architecture descriptions by means of meta-models for identifying change impacts. The approach relies on the following assumptions: (a) All artifacts of system development and operation must be considered. Focusing only on code is not sufficient. (b) Changes are initialized through evolution scenarios, resulting into predictable requirements on changes. (c) It is easier to identify the effort of fine-grained maintenance tasks, e.g. adding, deleting, or modifying architecture elements, than of coarse-grained maintenance tasks.

KAMP consists of two phases – *preparation phase* and *analysis phase* – and is followed by an *interpretation phase*. In the preparation phase, an architecture model for each design alternative to be compared is created. For this, meta-modeled architecture description languages are applied. Starting with a given evolution scenario, e.g. replacing a middleware technology or replacing the database, the considered change request(s) are identified by a human software architect. A change request among other things includes initially affected architecture elements, such as a particular software component or an interface, that are already known by the architect. In the analysis phase, artifacts affected by initially changed architecture elements are identified first. Then, lists of maintenance tasks (i.e. work plans) specific to the affected artifacts are created for each architecture alternative. This is performed automatically by the KAMP tooling for each architecture alternative and change request. In the interpretation phase, change efforts are estimated and architecture alternatives are compared by the architect based on the lists of maintenance tasks identified by the KAMP tooling. KAMP basically comprises three contributions [3]: (i) meta-models to describe system parts and their dependencies, (ii) a procedure to automatically identify system parts to be changed for a given change request defined manually as well as (iii) a procedure to automatically derive required maintenance tasks from a given change request to simplify the identification of a change effort and, by that, the maintainability estimation. Furthermore, KAMP is proposed to be applied for automated software project planning [2] and for deriving work plans to solve performance and scalability issues [1].

4 Applying KAMP to a CoCoME Evolution Scenario

Next, we describe how to apply KAMP to the CoCoME evolution scenario "replacing the database". Replacing a relational database by a non-relational database raises certain consequences. For example, because JDBC has just been developed to provide a connection to relational databases, the interface has to be replaced, too. KAMP is applied to identify such consequences of changing the database and to find out all affected components. While in the following only an overview is given, further details on the application of KAMP to the scenario and the affected architecture elements is available in [4].

In the preparation phase, the architect creates the architecture model of CoCoME and annotates it with additional information regarding building, deployment, and testing.

In the analysis phase, a copy of the current architecture model is created first. Second, the architect executes the structural changes in the architecture model. S/he deletes the old Database component in the architecture and adds another one. Moreover, s/he knows that the interface will not be usable any more and removes it and adds a new one. After the architect has marked all changes in the model copy, s/he triggers the calculation of the differences between the original and the copied model. The KAMP tooling recognizes that the Database component and its interface have been removed and another component and interface have been added. KAMP maps this information to maintenance tasks and builds up a first draft of a work plan which contains all tasks to realize the changes.

Next, possible side effects of changing single components are analyzed by investigating connections to other components. In the given scenario, the Data component (cf. [4]) is affected by changing the Database component. KAMP recognizes that Data is a composite component consisting of several sub-components which are included in the analysis and affected subcomponents are added to the work plan.

After all affected architecture elements have been mapped to tasks additional tasks for building, deployment, and testing are considered. For example, given that the architecture model has been enriched with test case information, for every test connected to the Data and Database component a *modify* and *run* task is suggested. This procedure results in comprehensive work plans suitable to implement and estimate changes.

5 Conclusion

We described the application of KAMP for change impact analysis in the new CoCoME evolution scenario "replacing the database". In the future, CoCoME will be further modified to create new and evolve existing artifacts by new evolution scenarios such as the introduction of mobile clients.

References

[1] C. Heger and R. Heinrich. Deriving work plans for solving performance and scalability problems. In *EPEW*, pages 104–118. Springer, 2014.

[2] O. Hummel and R. Heinrich. Towards automated software project planning - extending Palladio for the simulation of software processes. In *KPDAYS*, pages 20–29. CEUR Vol-1083, 2013.

[3] K. Rostami et al. Architecture-based assessment and planning of change requests. In *11th Intl. Conference on the Quality of Software Architectures*. ACM, 2015 (accepted to appear).

[4] T. Knapp. KAMP analysis applied to CoCoME. In *Seminar thesis, SDQ Chair, KIT*, 2012.

[5] M. M. Lehman and L. A. Belady, editors. *Program Evolution: Processes of Software Change*. Academic Press Professional, Inc., 1985.

[6] R. Heinrich et al. Integrating run-time observations and design component models for cloud system analysis. In *MRT*, pages 41–46. CEUR Vol-1270, 2014.

[7] S. Herold et al. CoCoME – the common component modeling example. In *The Common Component Modeling Example*, pages 16–53. Springer, 2008.