Reengineering of Component-Based Software Systems in the Presence of Design Deficiencies – An Overview

Markus von Detten and Sebastian Lehrig*

Software Engineering Group, Heinz Nixdorf Institute, University of Paderborn
Zukunftsmeile 1, 33102 Paderborn, Germany

sebastian.lehrig@uni-paderborn.de

Abstract

In reengineering, up-to-date architecture models are important artifacts to get an overview of a system and to plan and execute the necessary reengineering activities. If such models do not exist, software architecture reconstruction (SAR) techniques can be used to recover them from the system's source code. However, design deficiencies like Interface Violations can influence the architecture reconstruction and thereby adulterate the recovered architecture. This is currently not addressed by SAR approaches. Archimetrix is an iterative reengineering process that was developed in a PhD thesis at the University of Paderborn [5]. It aims at detecting and removing design deficiencies which influence the architecture reconstruction and thereby enable the recovery of more accurate architecture models.

1 Introduction

The maintenance of component-based software systems requires up-to-date models of their concrete architecture, i.e., the architecture that is realized in the source code. These models help in systematically planning, analyzing, and executing typical reengineering activities like bugfixing, extension, or modularization.

Often no or only outdated architectural views of such systems exist. Therefore, various reverse engineering methods have been developed which try to recover a system's components, subsystems, and connectors. However, these reverse engineering methods are severely impacted by design deficiencies in the system's code base, especially violations of the component encapsulation. As long as design deficiencies are not considered in the reverse engineering process, they reduce the quality of the recovered architectural views.

Despite this impact of design deficiencies, no existing SAR approach, e.g., SoMoX [2], explicitly integrates a systematic deficiency detection and removal into the recovery process.

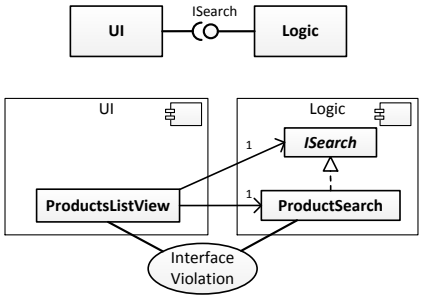

Figure 1: An example of the Interface Violation deficiency

Therefore, we have developed Archimetrix, a tool-supported architecture reconstruction process. It enhances a clustering-based SAR approach with an extensible, pattern-based deficiency detection. Archimetrix supports the software architect in detecting, ranking, and removing deficiencies and provides the means to preview the architectural consequences of such a removal.

We validated our approach on three case studies which show that Archimetrix is able to identify relevant deficiencies and that the removal of these deficiencies leads to an increased quality of the recovered architectures, i.e., they are closer to the originally designed architectures [3, 5].

2 Influence of Deficiencies on the Architecture Reconstruction

The upper part of Figure 1 shows an architectural view with the two components UI and Logic. They are connected via the interface ISearch. This could be a (very simple) architecture that the original system developer may have had in mind. The lower part of Figure 1 shows a view of the class distribution in these components. The UI component contains the class ProductsListView which accesses the interface ISearch in the Logic component. However, ProductsListView also directly accesses the class ProductSearch.

*The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant no 317704 (CloudScale).

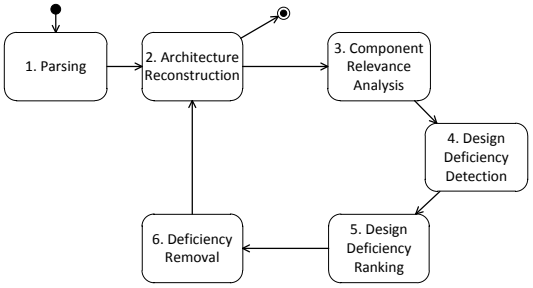

Figure 2: Overview of the reengineering process

This is, however, a neglect of Szyperski's principles of good object-oriented design which state that components may only communicate via their interfaces [4]. This *Interface Violation* deficiency leads to an increased coupling between the classes ProductsListView and ProductSearch (without the access between them, their coupling would be 0). Since coupling is one of the metrics that is frequently used by SAR approaches, this deficiency, thus, influences the recovery of the system's architecture. In the example, due to the high coupling between the classes, a SAR approach might reconstruct only one big component containing all three classes instead of two components.

3 Reengineering Process

Figure 2 visualizes the reengineering process with Archimetrix. Initially, the source code of the system is parsed into an abstract syntax graph (ASG) which represents the syntactic structure of the code.

In the second step, the system architecture is reconstructed from the ASG. The reconstruction is accomplished by a SAR approach that measures metrics like coupling or name resemblance for the system's classes and assigns them to components. Archimetrix uses the Software Model Extractor (SoMoX) which was developed at the Karlsruhe Institute of Technology for the architecture reconstruction [2]. The components in the reconstructed model follow the definition by Szyperski [4] and can be composed horizontally ("connected to each other") as well as vertically ("contained within each other"). This initial clustering can be influenced by existing deficiencies in the system.

In Step 3, we identify which of the reconstructed components are worthwhile inputs for the detection of deficiencies. These are, for example, components whose modification will probably influence the recovered architecture, i.e., components which are most sensitive to the removal of design deficiencies. This relevance analysis is also helpful because the detection of deficiencies in all components does not scale for larger systems.

Next, the detection is executed by matching predefined deficiency patterns in the ASG. The deficiency detection is accomplished by Reclipse which was also developed by our group [6]. Deficiency patterns are defined as object graphs which represent sub graphs of the ASG. The detection algorithm then tries to find occurrences of these object graphs in the ASG and annotates them accordingly. Deficiency patterns can be saved in a catalog so that they can be reused for a deficiency detection in other systems. The detection yields a list of deficiency occurrences in the previously selected components.

In the fifth step, the detected deficiency occurrences are ranked according to their severity. Deficiencies that have a stronger impact on the architecture are identified as the best candidates for a removal.

Finally, after the identification of the severe deficiency occurrences, the architect has to decide how the occurrences should be removed. This can either be done manually or by automated transformations that perform the reengineering in the ASG. If an automated transformation is available, the architect can also preview the architecture that would be reconstructed if the given deficiency occurrence was removed.

After the removal of one or more deficiency occurrences, a new iteration of the process can begin. The SAR step may now reconstruct a different architectural view as the removed deficiency occurrences can no longer influence it. The software architect can continue this cycle of architecture reconstruction, deficiency detection, and deficiency removal until he is satisfied with the reconstructed architecture.

4 Conclusion

In this paper, we gave an overview of the Archimetrix approach which enables the precise reconstruction of software architectures from source code by taking the influence of deficiencies into account. Archimetrix is currently used in the EU project CloudScale [1] where it is being extended in order to detect scalability-related deficiencies in cloud applications.

References

[1] G. Brataas, E. Stav, S. Lehrig, S. Becker, G. Kopcak, and D. Huljenic. CloudScale: Scalability Management for Cloud Systems. In *4th Int. Conf. on Performance Engineering*. ACM, Apr. 2013.

[2] L. Chouambe, B. Klatt, and K. Krogmann. Reverse Engineering Software-Models of Component-Based Systems. In *12th Conf. on Software Maintenance and Reengineering*, pages 93 – 102. IEEE, 2008.

[3] M. C. Platenius, M. von Detten, and S. Becker. Archimetrix: Improved Software Architecture Recovery in the Presence of Design Deficiencies. In *16th Conf. on Software Maintenance and Reengineering*, pages 255 – 264. IEEE, Mar. 2012.

[4] C. Szyperski, D. Gruntz, and S. Murer. *Component Software*. Addison-Wesley, 2nd edition, 2002.

[5] M. von Detten. *Reengineering of Component-Based Software Systems in the Presence of Design Deficiencies*. PhD thesis, University of Paderborn, Feb. 2013.

[6] M. von Detten, M. Meyer, and D. Travkin. Reverse Engineering with the Reclipse Tool Suite. In *32nd Int. Conf. on Software Engineering*, pages 299 – 300. ACM, May 2010.