From Architecture to Source Code – How to Ensure Architecture Compliance in the Implemented System[1]

Jens Knodel
Fraunhofer-Institute für Experimentelles Software Engineering (IESE),
Fraunhofer-Platz 1, 67663 Kaiserslautern, Germany
jens.knodel@iese.fraunhofer.de

Abstract

Software architecture is the key factor for efficient communication, planning, development, maintenance, and hence, the overall success of the development project. Architecting is an upfront investment made by development organizations to assure that the resulting system(s) will meet the required quality criteria in time and effort. Among others, the software architecture captures the envisioned structure of the system at development time (i.e., the decomposition of the system in manageable units like components). Verifying this planned decomposition late in the lifecycle of the software system reveals – too often – that the implemented system is not compliant to the specified structure. Consequently, efforts spent for architecting were made in vain because the decision and assumption made are no longer reliable and useful. To pro-actively prevent this structural decay, we propose constructive architecture compliance checking, which constantly monitors the modifications made by several (teams of) developers starting at day one of the implementation phase. Whenever structural violations are detected, the particular developer receives live feedback on the violations. Thus, a prompt removal of violations is possible, which ensures compliance of the implemented system with the architecture. Hence, the investments made into architecting are sustained over time.

Keywords: architecture compliance, constructive compliance checking, live feedback, reverse engineering, software architecture, SAVE LiFe, verification.

1. Introduction

Software architecture is the fundamental organization of a system embodied in its components, their relationships to each other and to the environment and the principles guiding its design and evolution [1]. Software architectures are typically documented by a set of architectural views whereby the most typical views are the conceptual view (i.e., the most abstract view capturing the fundamental concepts), the behavioral view (describing the system at run time), and the structural view (describing the system at development time) or named alike. The structural view describes the static decomposition of a system in terms of layers, subsystems, components, etc., and the relationships between the various elements. The structural view is the input for a coding process performed by one or several (teams of) developers. The coding process results in source code elements like folders, packages, compilation units, routines and dependencies between them, in other words, the source code elements refine the architecture on a lower level of abstraction. Practical experiences show that the realization – the implemented system – is typically not compliant to the specification – the planned structure (e.g., see [2]). More often than not, there are unwanted or unnecessary dependencies among the source code elements that violate the decision made on the abstract level of the architecture. Such structural violations constitute a severe threat to successful system development: on the one hand, structural violations contribute to long-term maintenance problems. The architecture as communication and decision vehicle is no longer reliable and useful, and the further evolution becomes time-consuming and effort-intensive because the abstractions provided by the architecture are no longer valid. On the other hand, the removal of structural violations causes a short-term overhead effort. Because they are detected late in the development process, they require additional, unplanned effort for workshops and meetings, refactorings, re-understanding of the source code, and regression testing to avoid unwanted side effects.

2. Idea

We propose structural compliance checking as a constructive quality engineering activity that pro-actively prevents the introduction of structural violations (opposed to analytical compliance checking that detects the

[1] This work was performed as part of the project ArQuE (Architecture-Centric Quality Engineering), which is partially funded by the German Ministry of Education and Research (BMBF) under grant number 01 IS F14.

violations late after their introduction). Constructive compliance checking is executed constantly and continuously for every single modification made to source code. Developers receive live feedback while they are implementing the system and violations are revealed as soon as they are introduced allowing their prompt removal. The high frequency of compliance checking (i.e., continuously executed all the time) turns the analytical technique into a constructive technique.

3. Tool Support

The constructive architecture compliance checking is automated as a variant of the SAVE tool (SAVE stands for Software Architecture Visualization and Evaluation). SAVE is a tool for analyzing and optimizing the architecture of implemented software systems. SAVE is a joint development of Fraunhofer IESE (Institute for Experimental Software Engineering IESE in Kaiserslautern, Germany) and Fraunhofer Center Maryland (Center for Experimental Software Engineering in College Park, Maryland, USA).

The new variant (called SAVE LiFe where LiFe stands for Live Feedback) is fully integrated into the Eclipse development environment (IDE) and enables the live feedback for multiple developers. The developers receive compliance checking feedback in real-time as soon as a central server performed the computation of the results on the modifications they just made. SAVE LiFe realizes the idea of constructive compliance checking as described above using the reflexion model technique [4]. It is a client-server system that supports two user roles, the architect and the developer. The architect defines the structural view and the specification of the mapping, while the developers' client triggers automatically fact extraction and computation of the compliance checking results on a central server. The compliance checking results are presented to the developers integrated in the source code editor (e.g., see Figure 1 showing a tabular list of violations and the source code editor highlighting one violations). Based on live compliance feedback on modifications just made, developer can immediately refactor the code, if necessary.

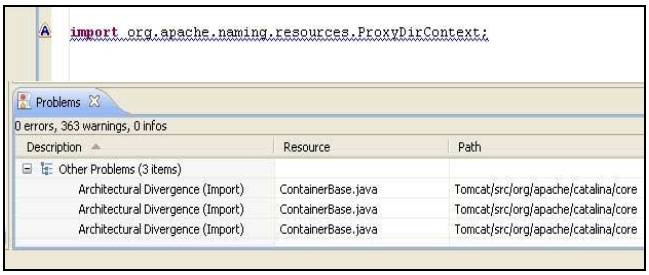

Figure 1: Violations Displayed in the Developers' Client

4. Application Experiences

First experiences in a controlled experiment [3] provided evidence that constructive compliance checking achieves superior results then regular development. The analysis results showed that the number of architecture violations was, after an initial peak, almost constantly 60% lower in the group that was supported by SAVE LiFe, compared to the control group. We observed that the number of architecture violations in the components developed by the SAVE-LiFe-supported teams was significantly lower than in those of the control teams. We consider this as an indicator for the positive impact of SAVE LiFe. Although there is a clear need for replication and further experimentation, the study provided some evidence regarding the benefits of constructive compliance checking. All teams invested approximately the same development effort to develop their components. The supported teams introduced less architectural violations, and, hence, when refactoring the implementation of the system to achieve architecture compliance, the supported teams would have about 60% less items to refactor. Consequently, the fewer refactoring items would eventually lead to effort savings. Thus, we claim that constructive architecture compliance checking has a positive effect on the implementation of software systems. We believe that the prompt removal of violations (as opposed to the late removal when applying analytical compliance checking) reduces the overhead effort. Our ongoing work addresses the further validation of constructive compliance checking in controlled experiments and industrial case studies.

5. References

[1] IEEE-Std.1471. ANSI/IEEE Std 1471-2000 - Recommended Practice for Architectural Description of Software-Intensive Systems. IEEE, New York, 2000.

[2] J. Knodel, M. Lindvall, D. Muthig, and M. Naab. "Static Evaluation of Software Architectures". 10th European Conference on Software Maintenance and Reengineering (CSMR), Bari, Italy, 2006

[3] J. Knodel, D. Muthig, D. Rost: Constructive architecture compliance checking — an experiment on support by live feedback. International Conference on Software Maintenance (ICSM), Beijing, China 2008.

[4] G. C. Murphy, D. Notkin, K. Sullivan: Software reflexion models: Bridging the gap between design and implementation. IEEE Transactions on Software Engineering, 27(4), April 2001.