Case Studies of Static Software Architecture Evaluations

Jens Knodel[1], Mikael Lindvall[2], Dirk Muthig[1], Matthias Naab[1]

[1] *Fraunhofer Institute for Experimental Software Engineering (IESE), Fraunhofer-Platz 1, D-67663 Kaiserslautern, Germany*
{knodel, muthig, naab}@iese.fraunhofer.de

[2] *Fraunhofer USA Center for Experimental Software Engineering Maryland(FC-MD), 4321 Hartwick Road, College Park, MD USA*
mikli@fc-md.umd.edu

Abstract

Architecture evaluations play an important role in the development and evolution of software systems since they determine how adequate the architecture is for its intended usage. This paper summarizes our practical experience with using static architecture evaluations and gives an overview on when and how static architecture evaluations contribute to architecture development. In particular, we identify ten distinct purposes and needs for static architecture evaluations derived from practical experiences in a set of industrial and academic case studies.

1. Introduction

One of the most important artifacts in the life cycle of a software system is the architecture [3] since it embraces the decisions and principles for the system to be developed. The goal of an architecture development method is to address such aspects and to provide the fundament for achieving organizational and business goals as well as meeting the functional and quality requirements of the system. To ensure the achievement of these goals, it is mandatory to integrate quality engineering activities into the architecture development method. This is especially true for software product lines since the product line architecture embraces the decisions and principles for each family member. Static architecture evaluations are a sound instrument for assessing and ensuring architectural quality without executing the source code, for instance by comparing the planned architecture against the implemented architecture.

In this paper, we report on ten distinct purposes and needs for conducting static architecture evaluations. The results of such an evaluation influence and determine subsequent steps in architecture development. The purposes and needs were derived from a set of industrial and academic case studies where we applied Fraunhofer PuLSE™ (Product Line Software Engineering, [1]) and Fraunhofer ADORE™ (Architecture- and Domain-Oriented Re-Engineering).

2. Static Architecture Evaluations

Static architecture evaluations analyze models of a software system without executing it. A special kind of static architecture evaluations is the analysis of the architecture conformance: a comparison of architectural models (the planned or intended architecture) against source code models (the actual or implemented architecture), as depicted in Figure 1 (based on the Reflexion model idea of [7] and [6]). Thereby, each model consists of a set of (hierarchical) model elements and different types of dependencies (variable accesses, calls, etc.) between them. The comparison requires a mapping (typically a human-based task) between the two models and assigns evaluation types to each dependency:

- Convergence – dependency exists in both the architectural model and the source code model
- Divergence – dependency is only present in the source code model
- Absence – dependency is only present in the architectural model

The architects can interpret the results by the total numbers of convergences, divergences and absences. In some cases, it is necessary to calibrate the evaluations (i.e., refinement of the architectural or the source code model, or the mapping).

Static architecture evaluations in general act as depicted in Figure 1, although not necessarily all of the three input artifacts of have to be available. The outcomes are documented in a results reports (graphical and textual), which can be processed further. We conducted a series of evaluations in the case studies (e.g., see [4]) and derived ten distinct purposes. All case studies were performed with SAVE (Software Architecture Visualization and Evaluation), an Eclipse plug-in developed at Fraunhofer IESE.

3. Purposes and Needs

Our architecture development method, PuLSE-DSSA [2], promotes scenario-based, incremental development in iterations to explicitly address stakeholders' concerns and view-based documentation to support the communication.

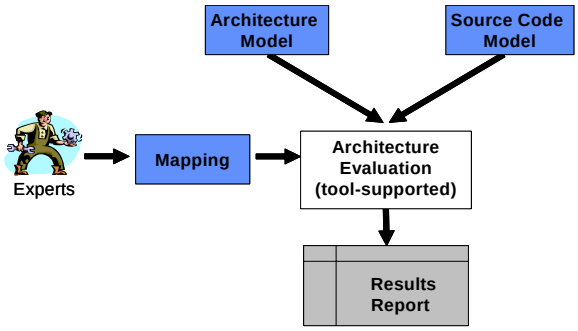

Figure 1: Static Architecture Evaluations

The main process loop of PuLSE-DSSA consists of four major phases: The **planning phase** defines the contents of the current iteration and delineates the scope of the current iteration. This includes the selection of a limited set of scenarios to be addressed in the current iteration, the identification of the relevant stakeholders and roles, the selection and definition of the views. In the **realization phase** solutions, patterns, and means are selected and design decisions taken in order to fulfill the requirements given by the scenarios. The **documentation phase** documents architectures in an organizational-specific set of views (relying on standard and customized views). The goal of the **assessment phase** is to analyze and evaluate the resulting architecture with respect to the achievement of business goals and for meeting functional and quality requirements.

We experienced the benefits of applying static evaluations within architecture development with PuLSE-in several case studies (e.g., [4], [5]). However, our experiences showed that such static evaluations differ in each of the four phases of PuLSE-DSSA in the following aspects: models (architecture, components, code), granularity, interpretation of the high amount of data generated, integration into further architecture development, impact on architectural decisions, and effort to spend on iterations and refinement. The differences in these aspects lead to ten distinct purposes (mapped to the four PuLSE-DSSA phases) which are the following:

- *Planning*:
  Product line potential: Analysis whether several existing systems comply to a single reference architecture although being separately developed and not yet constituting a real product line.
  Product alignment: Analysis to estimate the degree of required modifications in order to align an existing system to the product line architecture.
- *Realization*:
  Reuse potential: Analysis of the reuse potential of an existing architecture or a component in order to answer whether or not to reuse it.
  Component adequacy: Assessment of the adequacy of a component for the product line architecture and the component's internal quality.
  Comprehension: Assessment of program comprehension on a higher level of abstraction (i.e., an architectural level or a component level).
- *Documentation*:
  Consistency: Assessment of the consistency of documentation (e.g., architectural views, component engineering models) against the implementation.
  Completeness: Detection of elements that form part of an implementation, but are not yet documented.
  Re-documentation: Static decomposition (on a low level) to create high level architectural descriptions.
- *Assessment*:
  Evolution control: Monitoring the evolution of a system or a product line once released.
  Structure: Assessment of the structural decomposition of a system or a product line.

4. Conclusion

Static evaluations of software architectures are one good means to ensure the quality of software architecture. This work presents ten different purposes for conducting static evaluations and how these purposes are mapped into our architecture development method PuLSE-DSSA. The purposes and needs have been derived based on practical experience in industrial and academic case studies (e.g., [4]). Purpose and need determine how static architecture evaluations are applied, that is they act as an perspective on how to use the results for further architecture development.

5. References

[1] J. Bayer et al.: PuLSE: A Methodology to Develop Software Product Lines, 5th Symposium on Software Reusability (SSR'99), 1999

[2] J. Bayer et al: Definition of Reference Architectures based on Existing Systems, (IESE-Report 034.04/E), 2004

[3] IEEE Standard 1471 - Recommended Practice for Architectural Descriptions of Software-Intensive Systems, IEEE Computer Society, 2000

[4] J. Knodel et al.: "Static Evaluation of Software Architectures", European Conference Software Maintenance and Reengineering (CSMR), March, 2006

[5] R. Kolb et al.: Experiences with Product Line Development of Embedded Systems at Testo AG. Submitted to: The 10th International Software Product Line Conference (SPLC 2006), Baltimore, USA.

[6] R. Koschke, D. Simon: Hierarchical Reflexion Models, Working Conference on Reverse Engineering, 2003.

[7] 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.