Chapter 12
Tracing the Implementation of Non-Functional Requirements

Stephan Bode
Ilmenau University of Technology, Germany

Matthias Riebisch
Ilmenau University of Technology, Germany

ABSTRACT

A software architecture has to enable the non-functional properties, such as flexibility, scalability, or security, because they constitute the decisive factors for its design. Unfortunately, the methodical support for the implementation of non-functional requirements into software architectures is still weak; solutions are not generally established. Recently, there are only few approaches that actually deal with non-functional requirements during design; even fewer take advantage of traceability, which supports a mapping of requirements to solutions through the development process. Therefore, in this chapter the new architectural design method TraGoSoMa is presented, which supports these issues. The method uses a so-called Goal Solution Scheme, which guides the design activities, supports conflict resolution, decision-making, and the classification of solutions. For illustration purposes the chapter uses a case study from a reengineering project for a Manufacturing Execution System (MES) that is restructured according to the SOA principles and integrated with an Enterprise Resource Planning (ERP) system.

INTRODUCTION

Motivation

Performance, scalability, flexibility, security and other so-called non-functional requirements or quality properties are crucial for the success of nearly every software project. They bear even more risk than functional requirements, because they can hardly be implemented after making the major design decisions. The software architecture has to enable these non-functional requirements, because they constitute the decisive factors for its design. Several architectural methods emphasize the analysis of non-functional requirements (Hofmeister et al., 2000) and the design considering them (Bosch, 2000; Bass et al., 2002). Furthermore, in the field of requirements engineering

DOI: 10.4018/978-1-4666-1945-6.ch012

functional and non-functional requirements and their interdependencies are modeled using some mature and established goal-oriented approaches (Chung et al., 2000; Yu, 1995; Amyot, 2003). Therefore, some development steps of requirements engineering and architectural design are strongly related with each other. However, bridging the gap between these two research areas is still a critical issue (Galster et al., 2006) especially in the case when non-functional requirements change.

Service-oriented architectures are frequently applied for business-critical purposes. For these systems a long life expectancy constitutes an important concern, during which they have to be adjusted to a high number of changes to maintain their operation and their business value. Thus, evolvability is an important issue for this type of systems.

Traceability links support changes, and therefore the evolution of software systems, by expressing relations between artifacts in different phases of software development (Letelier, 2002). They facilitate program comprehension by expressing dependencies explicitly. They relate design decisions to constraints, and they are used to trace dependencies for checking the completeness of changes. These benefits can be achieved from fine-grained traceability links on the level of design elements and design decisions.

There is a considerable amount of research for managing traceability and for maintaining their accuracy during changes, which is for example discussed in (Mäder et al., 2006). However, the tracing of non-functional requirements usually requires a very high number of links, since typically a high number of a system's components depend on one such requirement.

There are some critical issues that make the whole process complex: for example (a) the mapping of high-level non-functional requirements to their low-level solutions and mechanisms, as well as (b) detecting and solving conflicts among non-functional requirements resulting in trade-offs, and further (c) the missing structuredness of existing methodologies or even their complete absence.

The management of traceability links, and with them the non-functional properties, requires a high human effort, first because of the high number of links and second because of missing rules inhibiting effective tool support. In order to reduce the effort for establishing, maintaining, and validating the traceability links, our work presents contributions to facilitate tool support, and hence, traceability of non-functional requirements, in several ways, which are described in the next sections.

Challenges

The proper treatment of non-functional requirements is a very important part of architectural design. Unfortunately, the methodical support for the implementation of non-functional requirements is still lacking even if this book addresses it. The absence of a consolidated set of solutions for non-functional requirements, the abstraction gap between requirements and design as well as the many influencing factors on design decisions reduce the applicability of detailed design instructions.

From the point of view of software architectural design there are only few methods that lead to an improved design process by especially considering non-functional requirements: for example the QASAR method (Bosch, 2000) or the Attribute-Driven Design (ADD) method (Bass et al., 2002). On the other hand, in requirements engineering there are adequate approaches for dealing with non-functional requirements in a goal-oriented way: for example the NFR framework (Chung et al. 2000), the i* framework (Yu, 1995), or the Goal-oriented Requirement Language (GRL) (Amyot, 2003). Although efforts are made to push these goal-oriented methods towards support for architectural design, and therefore some overlapping in the development steps can be observed, further research has to be done to bridge the gap.