Available online at www.sciencedirect.com

ScienceDirect

Procedia Engineering 9 (2011) 708–713

Engineering Procedia

TRIZ Future Conference 2006

TRIZ for software architecture

Daniel Kluender *

Embedded Software Laboratory, RWTH Aachen University

Abstract

A key element to designing software architectures of good quality is the systematic handling of contradicting quality requirements and the structuring principles that support them. The theory of inventive problem solving (TRIZ) by Altshuller offers tools that can be used to define such a systematic way. This paper describes the idea and preliminary results of using inventive principles and the contradiction matrix for the resolution of contradictions in the design of software architectures. By rearchitecting a flight simulation system these tools are analysed and their further development is proposed.

Keywords: Software architecture; Design principles; Patterns; Quality attributes; Contradiction matrix;

1. Introduction

While there's still some discussion about the definition of the term software architecture[2] it is commonly accepted to be defined as the structure or structures of a system, which comprise elements, the externally visible properties of those elements, and the relationship among them [2]. The process of designing a system's software architecture is shown in figure 1.

* Corresponding author.
E-mail address: kluender@informatik.rwth-aachen.de .
[2] See http://www.sei.cmu.edu/architecture/definitions.html for a discussion.

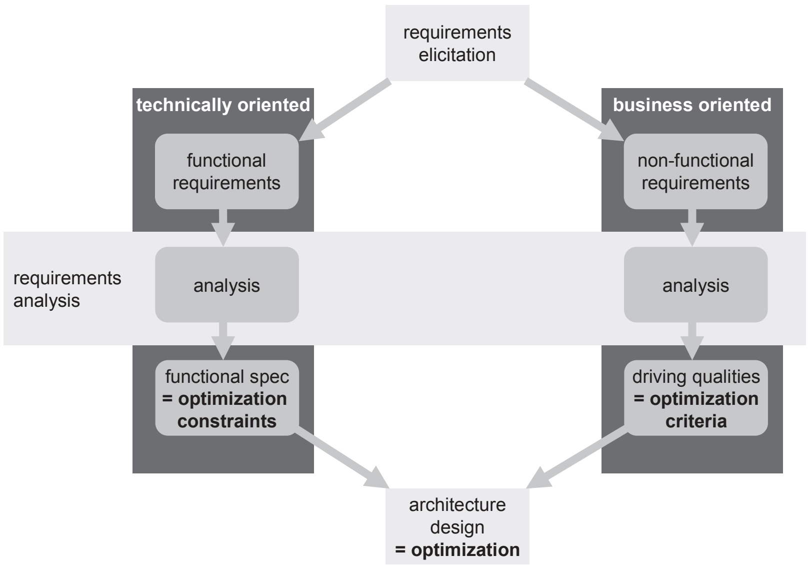
requirements elicitation
technically oriented
business oriented
functional requirements
non-functional requirements
requirements analysis
analysis
analysis
functional spec = optimization constraints
driving qualities = optimization criteria
architecture design = optimization

Figure 1. Two Paths of requirements analysis according to [2].

During requirements analysis functional and non-functional or quality requirements concerning a software intensive product are distinguished. Quality requirements are desired properties that surpass correct functionality like reliability, integrability, maintainability, testability or modifiability.

Non-functional requirements are frequently neglected because they are experientially harder to analyse but crucial for the success of software-intensive systems. Since the non-functional requirements are derivated particularly from a product's business goals they can not be analysed by a pure technically oriented inspection. During requirements analysis the functional specification is written down and the driving qualities are identified. Driving qualities represent the hard to implement but yet most important stakeholder interests in a product. Because of their important impact on the architecture the driving qualities are also called architectural drivers. Architecture design can be seen as an optimization problem with the driving qualities being the optimization criteria and the functional specification being the optimization constraints.

It has long been recognized that a system's software architecture has a major impact on the non-functional properties of a system like dependability, performance or modifiability [20]. Designing software architectures of good quality is therefore central to software engineering as is the evaluation of architecture quality.

Structuring principles which support certain qualities help the architect in finding the optimal architecture. These structuring principles can be architectural tactics or styles [2] like information hiding or architectural patterns [6] like a client-server architecture. These are generalized solutions for frequently occurring problems. Most structuring principles affect several qualities, either enabling or inhibiting them; e. g. information hiding supports the maintainability of a system but is impairing its performance. While the architect can choose from a set of well documented principles (see e. g. the work of Booch on a handbook of software architecture [4]) the merging and consolidation of different principles is by and large still an ad hoc and largely unsystematic process to date.

Conflicting quality requirements like performance and maintainability or conflicting structuring principles are compounding the design of a system's software architecture. The resolution of these conflicts relies heavily on the architect's experience and knowledge of the structuring principles. Software architecture represents the tradeoffs between the conflicting qualities that are acceptable for all stakeholders.

The systematic handling of contradictions between quality requirements or their according structuring principles can ideally result in the elimination or resolution of the conflict. The theory of inventive problem solving (TRIZ) by Altshuller et al. [1] can help to define such a systematic way. This paper describes the idea and preliminary results of using the TRIZ tools *inventive principles* and *contradiction matrix* for the resolution of contradictions in the design of software architectures. The rest of the paper is structured as follows: the next section gives a short