Requirements Engineering: Problem Analysis and Solution Specification (Extended Abstract)

R.J. Wieringa

University of Twente, the Netherlands
roelw@cs.utwente.nl

1 Introduction

Taken literally, the term "requirements engineering" (RE) is a misnomer. A requirement is something that is wanted; engineering, according to *Webster's*, is calculated manipulation. If our wants would arise by calculated manipulation, then something would be wrong. Our wants should not be engineered. What should be engineered, are solutions that meet our wants.

So what is requirements engineering? In this talk I discuss two views of RE, as problem analysis and as solution specification, and show that these two views meet in the discipline of IT systems architecting.

Architecture is central to web engineering because the web is an infrastructure for distributed coordination. Requirements engineering for web aplications therefore must deal with a distributed and sometimes fuzy set of stakeholders and with evolving requirements that will change once people use the application and explore new coordination mechanisms. In this context, requirements engineering is a distributed and concurrent process of problem analysis and solution specification.

2 Requirements Engineering and Problem Solving

The frequently heard mantra of software engineers is that requirements specify *what* a system should do, whereas a design says *how* it should do it. But the distinction between "what" and "how" is meaningless. We can ask *how* a system behaves externally and *what* its internal structure is, just as we can ask *what* its external behavior is and *how* it is structured internally. A better distinction is that between problem and solution. A *problem* is a difference between what is perceived to be the case and what is desired, that we want to reduce; a *solution* is an action that reduces this difference [1, 2]. One view of requirements is that they specify a problem; another view is that they specify a solution. I discuss both views in this talk.

Note that problem analysis and solution specification need not be sequentially related. In general, problem analysis and solution specification proceed jointly, with the engineer spending most, but not all of her time on problem analysis at

N. Koch, P. Fraternali, and M. Wirsing (Eds.): ICWE 2004, LNCS 3140, pp. 13–16, 2004.

the start of the process, and spending most, but not all of her time on solution specificatioon towards the end of the process [3, 4].

3 Requirements Engineering as Problem Analysis

If requirements specify a problem, then a requirements specification should describe

- what the problematic phenomena are,
- what the cause relationships between these phenomena are,
- by which norms these phenomena are problematic, and
- which stakeholders have these norms.

An example of this approach to RE is *goal-oriented RE*, which starts from a top-down analysis of user goals and refines these until desired properties of solutions are found [5–8]. Another example is the *problem frame* approach of Michael Jackson, in which frequently occurring problem structures are identified and are related to a frame concern, which relates the solution specification to a goal in the problem domain. A third approach sees problem-oriented RE as *theory-building*, in which a theory of the problematic phenomena is built, that will help us to specify a solution that takes away the causes of the problems [9]. All these approaches take a rather top-down approach in which the assumption is that users can specify their goals (even if these may be mutually inconsistent). In cases where users cannot specify their goals, yet other approaches are suitable, such as the task-support style of requirements specification proposed by Lauesen [10] or an evolutionary approach in which users are observed in their work, from which conclusions about their requirements are drawn [11, 12].

4 Requirements Engineering as Solution Specification

The view RE as solution specification is taken by the IEEE 830 standard [13] and by other authors on requirements [14, 15]. In this view, a requirements specification consists of

- a specification of the context in which the system will operate,
- a list of desired system functions of the system,
- a definition of the semantics of these functions,
- and a list of of quality attributes of those functions.

Classical techniques for software solution specification are structured analysis and, to some extent, object-oriented techniques [16, 17]. Note however that object-oriented techniques such as the UML are primarily oriented towards specifying the internal structure of software solutions, not towards specifying their requirements.

5 IT Systems Architecture

The two views of requirements come together in the concept of system architecture. I define an *architecture* as the set of relationships between the components of a system, that jointly ensure emergent properties of the system as a whole. The architecture of a building is the set of relationships between parts of the building that cause the building to have desired properties, such as room, shelter, functionality and appearance. In the same way, the architecture of a software system is the set of relationships between its components that cause the system to have desired properties, such as a desired functionality, behavior, semantics, and quality of service.

Architecture links requirements as problem characteristics to requirements as solution properties. Consider a set of problem-oriented requirements, that characterize a business problem, and a set of solution-oriented requirements, that specify desired solution properties. If part of this solution is a software system, then other parts may be novel work procedures and a new physical layout of an office. Thus, the solution has an architecture that is expected to solve the business problem and the desired software system plays a role in this architecture. And it is the architecture that links the software solution to the business problem. The same holds, *mutatis mutandis*, at lower levels of the aggregation hierarchy. For example, the component architecture of a software system links the component specifications to the software problem that they are intended to solve.

Architecture is the central problem in web applications because these applications should enable distributed coordination between people, and the architecture of these coordination mechanisms evolves by itself as well as is designed by people. As far as it evolves by itself, it may link unforeseen solution properties to unforeseen problems. If left to its own, evolutionary processes tend to deteriorate the architecture of a system. The challenge of requirements engineering for web applications is therefore to design architectures that enable this process.

References

1. Dewey, J.: How We Think: A Restatement of the Relation of Reflective Thinking to the Educative Process. D.C. Heath and Company (1933)
2. Gause, D., Weinberg, G.: Exploring Requirements: Quality Before Design. Dorset House Publishing (1989)
3. Cross, N.: Design cognition: results from protocol and other empirical studies of design activity. In Eastman, C., McCracken, W., Newstetter, W., eds.: Design Knowing and Learning: Cognition in Design Education. Elsevier (2001) 79–103
4. Wieringa, R.: Requirements Engineering: Frameworks for Understanding. Wiley (1996)
5. Antón, A., Potts, C.: The use of goals to surface requirements for evolving systems. In: International Conference on Software Engineering (ICSE'98), IEEE Computer Society (1998) 157–166
6. Dardenne, A., Lamsweerde, A.v., Fickas, S.: Goal-directed requirements acquisition. Science of Computer Programming **20** (1993) 3–50

7. Mylopoulos, J., Chung, L., Yu, E.: From object-oriented to goal-oriented requirements analysis. Communications of the ACM **42** (1999) 31–37
8. Yu, E.: An organization modelling framework for information systems requirements engineering. In: Proceedings of the Third Workshop on Information Technologies and Systems (WITS'93). (1993)
9. Wieringa, R., Heerkens, H.: Requirements engineering as problem analysis: Methodology and guidelines. Technical report, University of Twente (2003)
10. Lauesen, S.: Software Requirements: Styles and Techniques. Addison-Wesley (2002)
11. Beyer, H., Holtzblatt, K.: Contextual Design: Defining Customer-Centered Systems. Morgan kaufmann (1998)
12. Bondarouk, T., Sikkel, N.: Implementation of collaborative technologies as a learning process. In Cano, J., ed.: Critical Reflections on Information Systems–A Systemic Approach. Idea Group Publishing (2003) 227–245
13. IEEE: IEEE Guide to Software Requirements Specifications. In: Software Engineering Standards. IEEE Computer Science Press (1993) IEEE Std 830-1993.
14. Davis, A.M.: Software Requirements: Objects, Functions, States. Prentice-Hall (1993)
15. Robertson, S., Robertson, J.: Mastering the Requirements Process. Addison-Wesley (1999)
16. Wieringa, R.: A survey of structured and object-oriented software specification methods and techniques. ACM Computing Surveys **30** (1998) 459–527
17. Wieringa, R.: Design Methods for Reactive Systems: Yourdon, Statemate and the UML. Morgan Kaufmann (2003)