Requirements Traceability

Mirka Palo

Seminar Report

Department of Computer Science

University of Helsinki

30th October 2003

Table of Contents

1 Introduction

This paper is an introduction to seminar presentation about requirements traceability. In chapter 2 definition for requirements traceability is given. Reasons for requirements traceability in different phases of the system development are described in chapter 3. In chapter 4 a conceptual trace model is described and in chapter 5 two traceability reference models are described.

2 Definition

The following definition sums up the general view of the requirements traceability [Got94]:

"*The requirements traceability is the ability to describe and follow the life of a requirement, in both a forward and backward direction, i.e. from its origins, through its development and specification, to its subsequent deployment and use, and through periods of ongoing refinement and iteration in any of these phases.*"

3 Reasons for Requirements Traceability

The traceability needs of different stakeholders – project sponsors, project managers, analysts, designers, maintainers, and end users – differ due to differences in their goals and priorities. The requirements traceability is a characteristics of a system in which the requirements are clearly linked to their sources and to the artefacts created during the system development life cycle based on these requirements [Ram01].

In requirements engineering and elicitation phase it is important that the rationales and sources to the requirements are captured in order to understand requirements evolution and verification [Ram01].

Modifications during design appear e.g. if the requirements evolve or if the system is developed incrementally [Kne02]. During design phase requirements traceability allows to keep track of what happens when change request is implemented before a system is redesigned. Traceability can also give information about the justifications, important decisions and assumptions behind requirements [Ram01].

Test procedures can be traced to requirements or design and this kind of traceability helps to design and modify test procedures [Ram01].

Modifications after the delivery of the system will happen due to various reasons (e.g. to correct faults or to adapt the system to a changing environment). These kinds of modifications are called system evolution [Leh03]. Empirical studies show that even experienced software professionals predict incomplete sets of change impacts [Lin98]. With complete traceability, more accurate costs and schedules of changes can be determined, rather than depending on the engineer or programmer to know all the areas affected by these changes [Ram01].

4 Conceptual Trace Model [Kne01], [Kne02]

In [Kne01] and [Kne02] a change-oriented traceability for embedded systems is studied and the results of the impact analysis approach consist of three parts:

- fine-grained trace model
- set of process descriptions that describe how to establish traces and how to analyze the impacts of changes
- tool support that provides (semi-) automatic impact analyses and consistency checking of implemented changes.

The trace model determines the types of documentation entities and relationships to be traced to support impact and implementation of system requirements changes. The conceptual trace model consists of conceptual system model and conceptual documentation model. These models are described in more detail in the subchapters below.

4.1 Conceptual System Model

A conceptual system model describes logical entity types and their dependency and refinement relationships included in a software system at different abstraction levels. Logical entity types and their relationships depend on the investigated application domain. The conceptual system model distinguishes mainly between types of:

- Items at different abstraction levels (e.g. controlled environmental item)

- Tasks at different abstraction levels. Each system task must be related to a set of dependency relationships between monitored environmental items and one controlled item.
- Dependency relationships between entity types at one abstraction level (e.g. monitored items must have influence relationships to controlled items).
- Refinement relationships between entity types at different abstraction levels (e.g. a system task must have a refinement relationship to a software task and at least two hardware tasks).

4.2 Conceptual Documentation Model

A conceptual documentation model describes representation entity types included in different software documents on various levels of abstraction of a software system and their relations. Besides the dependency and refinement relationships that can be taken from the conceptual system model, the conceptual documentation model includes representation relationships. Representation entity types and their relationships depend on the product model and description techniques chosen. The documentation model extends the description elements to allow unambiguous identification of each logical entity type. It distinguishes mainly between types of:

- Documentation entities (e.g. system use cases of a system use case diagram, or software design methods of a software design class diagram).
- Dependency relationships between documentation entity types at the same abstraction level (e.g. each system use case must have an influence relationship to an influenced actor). These relationships are derived from the conceptual system model. Each relationship described for a logical entity type must be true for a documentation entity type that represents the logical entity type.
- Refinement relationships between documentation entity types at different abstraction levels (e.g. each system use case must have a refinement relationship to a software use case). These refinement relationships are derived from the conceptual system model.

- Representation relationships between documentation entity types that represent the same logical entity type (e.g. a use case must have a representation relationship to a use case description because both represent a system task.

5 Traceability Reference Models [Ram01]

Reference models in general are prototypical models of some application domain. The purpose of reference models is to significantly reduce the task of creating application-specific models and systems: The user selects relevant parts of the reference model, adapts them to the problem at hand, and configures an overall solution from these adapted parts. Since the analysis of a domain can take an enormous effort when started from scratch, the use of reference models has been reported to save up to 80 percent in development costs for systems in standardized domains [Sch98].
The reference models described in [Ram01] are based on several empirical studies. The data collection spanned a period of over three years. The main study comprised 30 focus group discussions in 26 organisations, which were conducted in a wide variety of industries including e.g. defence, aerospace, pharmaceuticals, electronics, and telecommunications. The participants had an average of 15.5 years experience in several key areas of systems development including e.g. software engineering, requirements management, software testing, system integration, systems analysis, maintenance, and software implementation.
While doing the study it became apparent that the participants could be categorized into two distinct groups with respect to their traceability practise. These groups are referred as low-end and high-end traceability users and there are separate reference models for these groups. These models are described in the subchapters below.

5.1 Low-End Traceability Model

The low-end traceability users have the following characteristics:

- The typical complexity of the system is about 1000 requirements
- The traceability experience level is from zero to two years
- The user definition of traceability is the documents transformation of requirements to design

- The main applications of traceability are requirements decomposition, requirements allocation, compliance verification and change control.

The low-end traceability model can be seen in the figure 1 below.

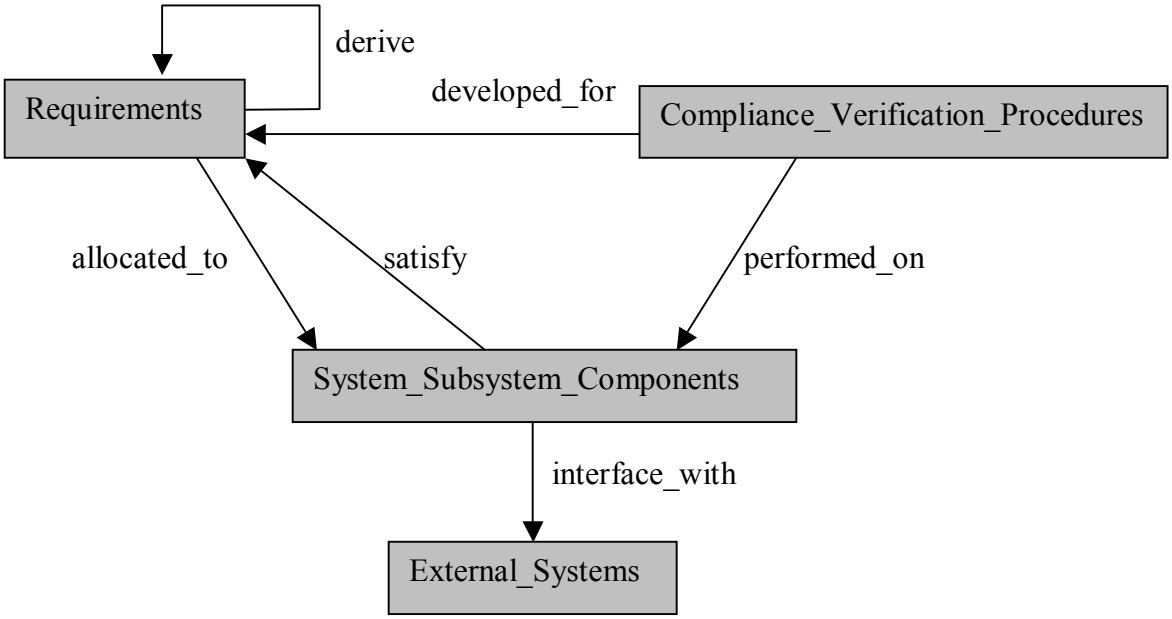

Figure 1: Low-end traceability model

Typical low-end users view requirements traceability as providing a link from initial requirements to the actual system components that satisfy those requirements. Lower level refined requirements are derived from higher level system requirements. Original and derived requirements are allocated to system components. By capturing which components satisfy various requirements and which requirements are mapped to different components, the designer is able to verify that all requirements are addressed by the system. In the compliance verification phase of systems development, low-end users use the requirements database, which contains the most current version of the system's validated requirements, to develop the system compliance verification procedures such as tests or simulations. If a change should occur in the requirements, then the traceability links could identify the compliance verification procedures that must be modified or redeveloped. Compliance verification procedures are performed on the system

components verifying that the component satisfies the requirements. Results of the tests are used to verify that the system works and that it meets all of the requirements. A system component may depend on others and may also interface with external systems. This information is used in evaluating how a requirement is satisfied by a system component.

Low-end users lack especially in the area of capturing rationale, see e.g. the following quote from a user: "*Often we have no idea who made these decisions, and how they impact the rest of the effort. Simply trying to do these at the end of the project or after the fact does not work. Often the people who worked on it are gone without a trace, of what happened… not disciplined enough to document these… with all the demands on the team.*"

5.2 High-End Traceability Model

The high-end traceability users have the following characteristics:

- The typical complexity of the system is about 10000 requirements
- The traceability experience level is from five to ten years
- The user definition of traceability is that it increases the probability of producing a system that meets all customer requirements and will be easy to maintain
- The main applications of traceability are full coverage of life cycle including user and customer, capturing discussion issues, decision and rationales, capturing traces across product and process dimensions.

High-end users of traceability employ much richer traceability schemes than low-end users and also use traceability information in much richer ways. Therefore the high-end model is divided into four parts for clarity:

- Requirements management submodel
- Rationale submodel
- Design allocation submodel
- Compliance verification submodel.

These submodels are described in the subchapters below.

5.2.1 Requirements Management Submodel

With the requirements management submodel the requirements can be traced throughout the lifecycle to provide stakeholders with a view to understand and evaluate whether system supports critical success factors. The requirement management submodel is shown in figure 2 below.

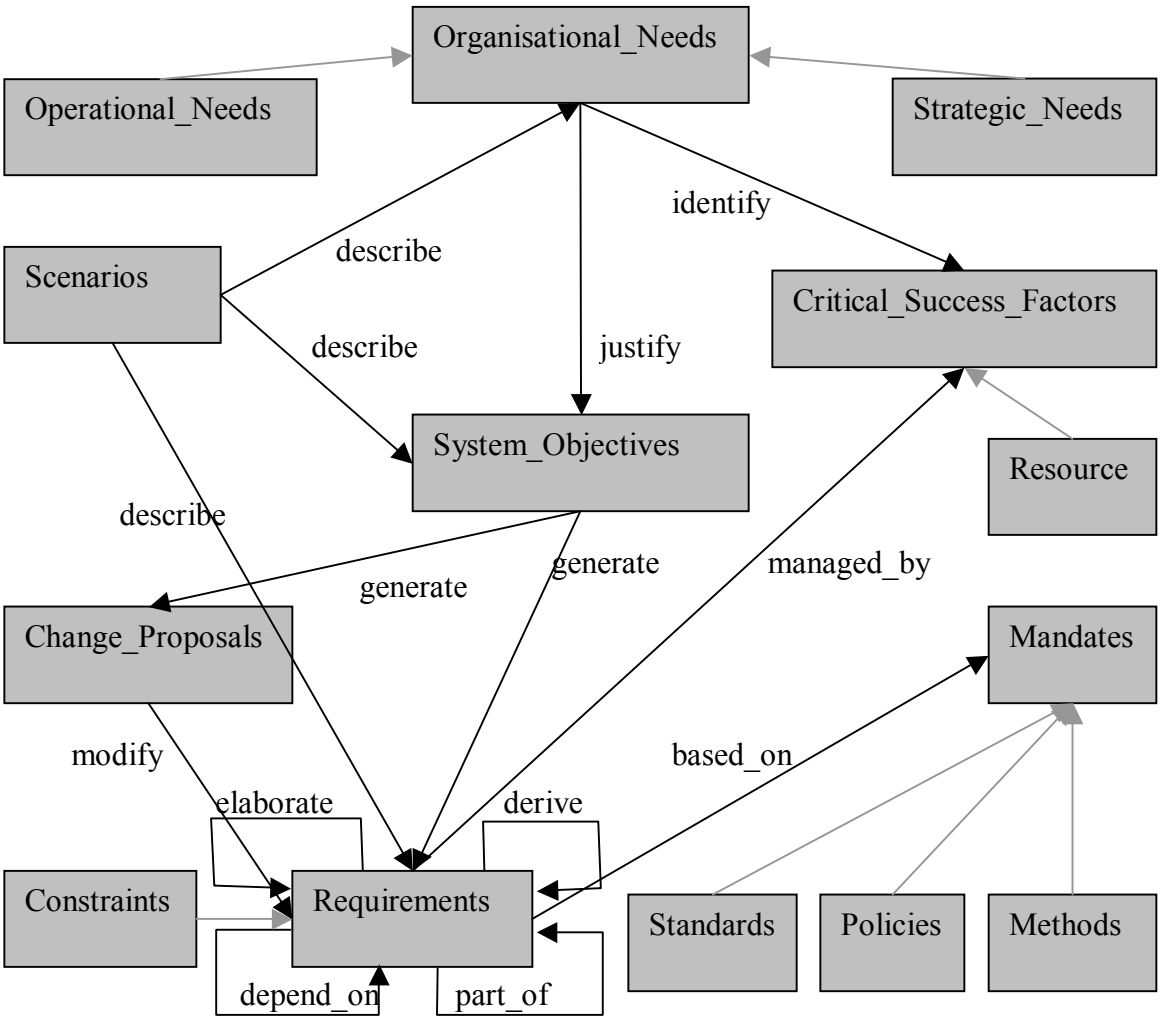

Figure 2: Requirements management submodel

The requirements management submodel takes into account the following issues:

- systems are built to satisfy organizational needs
- organizational needs are detailed in scenarios

- system objectives are justified by organizational needs, stakeholders specify the system objectives
- requirements are generated from system objectives
- organizational needs (i.e. stakeholders) identify critical success factors, e.g. resources can be one of the critical success factors
- requirements for the system are managed by critical success factors
- requirements may also be based on standards, policies and methods
- constraints may be treated as a type of requirement
- lower level requirements are derived from higher level requirements
- some requirements are elaborated by others, providing further explanation or clarification
- requirements also depend-on others
- complex requirements are often broken down into their components, identifying simpler requirements that form a part-of them.

5.2.2 Rationale Submodel

The rationale submodel maintains the information about how decisions are made to resolve issues or conflicts throughout the system lifecycle to ensure that customer requirements are understood and satisfied. The rationale submodel takes into account the following issues:

- objects (e.g. components, requirements, designs) generate issues or conflicts
- issues are resolved by decisions
- decisions may affect requirements
- various alternatives that address the resolution of issues are considered
- arguments for and against each alternative may be proposed
- decision to select one or more alternatives id often influenced by the critical success factors
- assumptions underlying the various components of the deliberation are also recorded

5.2.3 Design Allocation Submodel

The design allocation submodel shows the relations between requirements and design components. The design allocation submodel takes into account the following issues:

- requirements drive design
- design is often based on mandates (e.g. standards, policies or methods) that govern the system development activity
- system or subsystem components are the building blocks of the system and they are defined or created by the design process
- requirements are allocated to components that are supposed to satisfy them
- components depend on other components
- components can be part-of other components
- resources are used by components
- functions are performed by components
- functions are addressed to requirements
- components depend on external systems.

5.2.4 Compliance Verification Submodel

The compliance verification submodel is used to certify completeness and correctness of the system and identify changes that may be necessary to meet the objectives. The compliance verification submodel takes into account the following issues:

- the development of compliance verification procedures (e.g. prototyping, simulation, testing and inspection) is governed by their use of resources
- mandates (e.g. standards, policies or methods) are commonly the basis of compliance verification procedures and determine which procedures are required and how they are to be performed
- compliance verification procedures either verify how the components satisfy requirements or help generate change proposals for requirements, or design or implementation.

6 References

[Got94] Gotel, O., Finkelstein, A.
An Analysis of the Requirements Traceability Problem
Proc. of First International Conference on Requirements Engineering, 1994, pages 94-101

[Kne01] von Knethen, A.
A Trace Model for System Requirements Changes on Embedded Systems
Proc. of 4th International Workshop on Principles of Software Evolution, September 2001

[Kne02] von Knethen, A.
Change-Oriented Requirements Traceability. Support for Evolution of Embedded Systems
Proc. of International Conference on Software Maintenance, October 2002, pages 482-485

[Leh03] Lehman, M., Ramil, J.
Software Evolution – Background, Theory, Practice
Information Processing Letters, Vol. 88, Issues 1-2, October 2003, pages 33-44

[Lin98] Lindvall, M., Sandahl, K.
How well do experienced software developers predict software change?
The Journal of Systems and Software 43, 1998, pages 19-27

[Ram01] Ramesh, B., Jarke, M.
Toward Reference Models for Requirements Traceability
IEEE Transactions on Software Engineering, Vol. 27, No. 1, January 2001, pages 58-93

[Sch98] Scheer, A.
Business Process Engineering: Reference Models for Industrial Enterprises
Springer-Verlag, 1998