An Introduction to Requirements Traceability

Roel Wieringa[1] [2]

November 1, 1995

[1]Faculty of Mathematics and Computer Science, *Vrije Universiteit*, De Boelelaan 1081a, 1081HV Amsterdam. Email roelw@cs.vu.nl.

[2]Partially supported by Esprit Project 2RARE (2 REal Applications of Requirements Engineering), contract number 20424.

Abstract

This report surveys the requirements traceability literature and gives some recommendations for further research and for an approach to consultancy concerning traceability in the 2RARE project.

The problem of maintaining traceability in a development project is viewed as the problem of maintaining an information system that maintains the relevant links between items developed during the process. These items may vary from requirements to design and implementation documents. To develop and implement such an information system, we must identify the needs for tracking information, made a model of this information, design usage procedures for the system, and implement the system. Accordingly, the literature on traceability is organized under the headings *needs*, *models* and *usage*. Furthermore, tracking *tools* are briefly reviewed.

Acknowledgements: This report benefited from input given by Eric Dubois, Anthony Finkelstein and Hanna Luden.

Contents

Chapter 1

Introduction

A **requirements specification** is a document that describes (1) the objectives that a product must satisfy and (2) the observable behavior that the product must have in order to fulfill these objectives [41]. The **objective** of any product is to satisfy a need in its environment. At the top level, the product objectives are a specification of the needs that exist in the product's environment. This can be the result of a ISAC change analysis (the product objectives are to solve a number of user problems), of a business strategy analysis (the objectives are to implement a number of business strategies), etc. The most general way in which a product satisfy its objectives is laid down in a **product idea**, which is a general, one-sentence description of what the product does. The product idea can be specified more detailed as a number of required **product functions**, which are groups of required product behavior. **Product behavior** consists of the interactions of the product and its envronment.

In this report, the observable behavior is taken to include required system functions as well as nonfunctional properties. All required properties, if they are to be observable, concern some aspect of observable system behavior. A requirement may be represented as a piece of text, a table, a figure reference, or even a piece of music (e.g. representing a soothing melody to be played when a deadline passes.)

The role of a requirements specification in the product life cycle is to serve as a means of communication between different stakeholders in the product, such as the customer or marketing department, developers, and maintenance staff:

- The developer and customer use the specification to agree on the functionality that the product is going to have (including product objectives and functional and nonfunctional requirements);
- customers use it (ideally) to derive acceptance tests for the product;
- different developers use it to agree on interfaces between the parts of the product they work on;
- maintenance staff use it as a norm against which to correct errors in the product;
- customers and maintenance staff use it to agree on which changes in the product best reflect changed desires of a user or of the market.

In such a mature use of requirements specifications, requirements continue to play a role throughout the product life cycle and the distinction between using them for initial development or for adaptive maintenance has disappeared. Two crucial features in this use of requirements are:

- Requirements specifications act as a means communication among stakeholders with sometimes conflicting interests;

- Requirements specifications must continuously be updated to reflect changes in users' desires;
- requirements specifications may have to be updated because of problems encountered in implementation.

In order to fulfill this role, the requirements specification is developed along with other deliverables, such as design documentation and the finished product. At the end of initial development, the specification, design and implementation must agree with each other and throughout all subsequent changes, all documents must be kept in agreement with the changes in user's desires and in the product. To realize this, all parts of all documents that must change together, must be linked together, as well as to the user's desires and to parts of the finished product. The documents created and maintained during the initial development and throughout the life cycle of the product are called **traceable** if each part of each document can be traced to any other part of any relevant document in such a way that parts that must change together, are linked to each other. The linked documents may describe user needs, system requirements, system design, system implementation, system context, etc.

There is no unique definition of traceability. The above definition is a generalization of the definition given in the IEEE Guide to Software Requirements Specifications [19]:

> A software requirements specification is traceable if the origin of each requirement is clear and if it facilitates the referencing of each requirement in future development or enhancement documentation.

In a traceable requirements specification, different specification components are linked if they impact each other, so that a change in one requires a change in the other. Surveying the definitions of traceability given in the literature [7, 12, 13, 21, 30], the following definitions give a consensus:

- **Forward traceability** of the requirements is the ability to trace components of a requirements specification to components of a design or of an implementation.
- **Backwards traceability** of the requirements is the ability to trace a requirement to its source, i.e. to a person, institution, law, argument, etc. that demands the requirement to be present.

The concepts of forward and backward traceability are applicable to other deliverables of development as well. For example, **Backward traceability** of the implementation is the ability to trace an implementation component to the requirements components that it must satisfy.

The problem of realizing traceability in product development is the problem of implementing and using a **tracking information system** containing all relevant development information. The users of this system are all stakeholders in the development process: sponsors, customers, developers, users of the product, and managers of the development process. Implementing the tracking information system, we must identify the needs of the stakeholders, define a model of the the data to be stored in it, and define procedures for system usage. We use this viewpoint as ordering principle for the literature survey in this report. Chapter 2 lists the tracing needs of the stakeholders as identified in the literature. Chapter 3 surveys the models of the development process that have been proposed for traceability tools. Chapter 5 surveys what has been said about the management of traceability, i.e. about the use of the tracking information system. Chapter 4 surveys current software tool support for traceability. Chapter 6 contains a discussion and conclusions.

Chapter 2

The need for traceability

Based on a survey of the literature, on focus group interviews and on empirical studies of development projects, Ramesh et al. [35, 36] list the following benefits of traceability. Below, these are grouped according to the needs of stakeholders in the development process.

- **Project management**. When requirements are linked to the components that implement them, then:
  - The impact of a change in requirements can be estimated;
  - Conflicts between requirements can be discovered earlier (and unexpected product delays avoided);
  - Requirements not yet satisfied by the implementation can be collected, and the work to be done to satisfy these remaining requirements can be estimated;
  - Future systems will have reduced development time and effort, because past implementation decisions can be reused.

  In general, a tracking information system helps project management to track project status.
- **Customer**. If the tracking information system records which requirements are satisfied by which parts of the implementation, and which tests must be performed to ascertain the presence of a requirement, then
  - The quality of the product with respect to the user requirements can be evaluated;
  - It can be ascertained that (costly) goldplating has been avoided, because all components of the implementation can be traced to at least one requirement;
  - Acceptance testing can refer directly to the user requirements being tested for.
  - User requirements are linked to design requirements, so that development personnel can keep their focus on the user requirements that they are trying to satisfy.
- **Designer**. The tracking information system should record the results of design, the justification of the results, alternatives considered, and the assumptions made in a decision. If these are recorded alongside with the links between the requirements and the design, then:
  - The designer can more easliy verify that a design satisfies the requirements;
  - The designer can estimate the impact of a change in requirements on the design;

- The designer can understand the reasons why a certain design was accepted and another rejected even when the design was produced long time ago by a designer not present anymore (these reasons may relate design decisions to nonfunctional requirements;
- The designer can estimate the impact of a change in available implementation technology on the design assumptions and hence on the design alternatives.
- The designer can reuse design components in other projects, because the assumptions under which the component will work are recorded.

The tracking information system is a kind of "corporate memory" that can be used to speed up decision making in future development projects.

- **Maintainer**.
  - Estimate impact of a change in requirements on other requirements (discovery of conflicts, dependencies);
  - Estimate the impact of a change in requirements on the implementation;
  - Estimate permissibility of change in implementation with respect to (unchanged) requirements requirements.

Chapter 3

Models of traceability

Sections 3.1 to 3.4 review the ESE, PRISM and NATURE models of the development process and the model developed for the Department of Defense. Section 3.5 lists the kinds of links that can be maintained between items. In section 3.6, some general conclusions are drawn.

3.1 The evolution support environment (ESE) system

The ESE system [32] is a research product built to support the evolution of software artifacts, called *objects*. Each object has many *versions* and different objects can be *linked*. There is a *generic link* between objects when most (!) versions of the object are linked. Objects are organized hierarchically into *modules*, *collections of modules*, and *members* (consisting of collections of modules). An example of a member would be the Ingres system version 1.3. The system provides reporting facilities, which includes reporting about links between objects. Three kinds of links are distinguished:

- **Hierarchical links** between objects at different levels of the hierarchy.
- **Historical links** between versions of one object.
- **Development links** between different objects at different stages of development.

ESE is implemented using Ingres and the SCCS version control system.

3.2 The PRISM model of changes

The Prism model of changes [27] consists of a *dependency structure*, containing change items, and a *change structure*, containing changes. Change items are classified as people, policies, laws, software development processes, resources (including time), and results produced by processes. These are organized in the following layers: individual, group, project, organization, nation, community. The dependency structure maintains links between change items. Per layer, change items are organized in groups of tightly connected items called *sheets*. For each sheet, one can specify a *sheet program* that specifies the changes to other sheets when this sheet changes.

The change structure records changes. For each change are recorded: the type of the changed item, location of the item in the environment, feedback concerning the results of the change, properties of the change, substructures of the change. There is a long list of change properties, including the following:

- Rationale of the change: source, decision, advantages and disadvantages, reason.

- Type of change: static or dynamic item, corrective/adaptive/perfective
- Size of the change: impact, cost of obeying, cost of not obeying.
- Structure of the change
- Reliability of the change process
- Person responsible for the decision
- Phase of the project
- Anticipation of the change

The full reports upon which this paper is based may be interesting, because they also survey the literature [24, 25, 26].

3.3 The NATURE project

The NATURE project is described by Pohl, Doemges and Jarke [30] and by Jarke, Pohl, Rolland and Schmitt [21]. In the NATURE project, requirements engineering is viewed as a process that moves through a space with three dimensions. Correspondingly, there are three dimensions to a trace of the requirements engineering process. These are maintained by a traceability tool called PRO–ART:

- **Representation dimension**. This ranges from informal to formal. Moving along this dimension is mostly a technical problem. Informal text is manipulated by a hypertext editor, semiformal representations (ER or DF diagrams) by graphical editors. Hypertext nodes can be linked to diagram components.
- **Agreement dimension**. This ranges from partial to complete. Moving along this dimension is a social process. This is represented by *issues* about which a *decision* must be made. An issue may be related to an object in the specification dimension. About each issue, one or more *positions* are stored, and for each position, the *arguments pro* and the arguments *contra* are recorded. Decisions can be revised, introducing *revised decisions*.
- **Specification dimension**. This should perhaps be called *understanding*, for it ranges from opaque to complete understanding of the specification. Moving along this dimension is a psychological process. A model has been made of over 30 different kinds of requirements with about 100 attributes, organized in a taxonomic hierarchy. A tool is under development that can instantiate this to any of the known requirements specification standards (IEEE–830, DOD–2716A, etc.). This is orthogonal to the other two dimensions, for each requirement in a standard can range from informal to formal and there can be partial or complete agreement about it.

The intention is that recording traces of the requirements engineering process will be used for process improvements.

3.4 The DoD model

In a series of reports [10, 33, 35, 36, 34], Ramesh and others develop a model of the development process that represents traceability information.[1] The model is based on interviews with focus groups,

[1] I did not receive [15, 23].

protocol analysis of developers at work, and one-on-one interviews of members of a particular development project. The resulting model is presented in [35]. It records links between requirements, design or implementation components, system objectives, organizational needs, stakeholders, decisions, assumptions, alternatives, verification procedures (inspections, tests, prototypes, simulations), standards, and many other kinds of items. It is the most ambituous traceability model currently presented in the literature. A simple version of the model is presented in [36].

3.5 The semantics of links

To give an impression of the kinds of links that can be represented, I give a list of links that have been proposed in the literature:

- Linking a requirements to its origin:
  - A requirement can be linked to its **source document** [9, 11].
  - A **requirement** can be linked to its **source**, which in this case is a role played by a person. The link may be that the source is documentor, author, or principal (the responsible person) of the requirement [12, 13].
  - A design element can be linked to its designer, to the person who validated it, or to the person(s) who modified it [33].
  - A **requirement** can be linked to its **justification** [21, 30, 31], which gives the reasons for the requirement.
  - A **requirement** can be linked to its **explanation** [42], which explains the meaning of the requirement.
  - In QFD, a **customer attribute** can be linked to a **design attribute** [5, 16, 38]. The strength of the link indicates the degree in which the customer attribute is supported or violated by the design attribute.
- Linking a requirement to artefacts that arise during development:
  - Linking **system requirements** to **subsystems**: Dorfman [8, page 9] and Davis [7, page 192]. The meaning of a link is that the requirement is realized by the subsystems to which it is linked. This is called **allocation** of requirements to subsystems.
  - Linking **system requirements** to **requirements** of the subsystems to which the requirement has been allocated: Dorfman [8, page 9] and Davis [7, page 192]. The meaning of a link is that the system requirement is realized if the subsystem requirements are satisfied. This is called **flowdown** of system requirements to subsystem requirements.
  - Design attributes can be traced onwards to component (i.e. subsystem) attributes, process planning attributes, production attributes, etc.
  - Design attributes can be linked to each other, to indicate strength of coupling or strength of conflict.
  - A **test** can be linked to a **requirement** [20].
- Linking parts of a specification at the same level of aggregation:
  - Informal, semiformal, and formal parts of the specification can be linked.

- A system requirement can be linked to a **more detailed** requirement [9, 11] on the same system. For example, a system requirement can be a more detailed specification of an aspect of a more abstract requirement on the same system. This is a movement from less to more detail of external behavior required of the same system, while flowdown is a movement from systems to subsystems.
- Different parts of **source code** can be linked.

This list has been assembled from the traceability literature. Looking at CASE tools, we can find still more links:

- Links between different parts of a requirements model (e.g. between different levels of a hierarchicah DFD, or between entity types in an ER diagram and data stores in a DFD).
- Links between different parts of software (calling hierarchies, import hierarchies, etc.).
- Dependency links between software (application routines that depend upon particular system software, graphical routines that presuppose certain hardware, etc.).

Turning to system development methods, we find still more links:

- Requirements can be linked to the problems they are intended to solve (ISAC).
- Problems can be linked to problem owners (ISAC).
- Requirements can be linked to critical success factors (Information Engineering & many other methods).
- Information needs can be linked to managers (Information Engineering).
- Required functions can be linked to the business mission (function decomposition tree).

These lists are not exhaustive.

3.6 Discussion

The above surveys make clear that there is as yet no agreement on a model of the development process and that there is no limit to the number of links that we can add to the model. The ultimate traceability tool is the world itself. To get a useful model of the development process, the model must match the needs of the developers with the resources available to maintain the traceability information. The traceability effort must also be related to the maturity level of the development organization [18]. The best strategy to implement traceability in a development organization seems to be to start with a simple model of the development process, so that developers need to spend relatively little effort to maintain the links, and incrementally extend this according to the needs of the developers. This strategy agrees with Humphrey's [18] strategy of implementing configuration management in the organization.

Chapter 4

Software support for traceability

Section 4.1 mentions a few CASE tools that offer support for traceability, but whose main functionality lies elsewhere (e.g. in diagram editing or code generation). Section 4.2 surveys software tools whose main functionality is the support of traceability. Section 4.3 summarizes the techniques used for representing links. Section 4.4 discusses the state of tools support.

4.1 CASE tools

Some CASE tools provide traceability features. For example, Maestro [37] is a CASE tool with diagram editing and code generation facilities, that also contains a facility to read COBOL applications and produce cross-references between different parts of the programs, that can be used for an analysis if impact of changes. RDD-100 is a requirements specification too with diagram editing features (ER, DFD, SADT) that supports traceability between different diagrams.

4.2 Traceability support tools

4.2.1 ARTS

ARTS system [9, 11] is a database built to maintain requirements links. It can produce cross-reference tables in various formats, including the matrix form. The system was developed in-house at Lockheed and there is some eight years experience with using it.

4.2.2 RTM

RTM [17] is a requirements management and traceability system implemented in an Oracle database system, with a read-only interface to the teamwork and STP CASE tools. All requirements documents are stored in the databse with document identity and paragraph numbers. Requirements can be categorized using keywords. Requirements can be refined by creating a *substitutes* link between the refined and the unrefined requirements. RTM can read Real Time Structured Analysis models fro Teamwork or STP. The analyst can link the components of these models to the requirements. In addition, tests for requirements can be specified, stored in the database, and linked to the requirements. When requirements are changed, RTM can determine which other objects in the database are likely to change.

4.2.3 RADIX

Yu [42] describes the RADIX system, developed at AT&T to support the development of a complex switch development environment. RADIX is a Troff macro package to format requirements, define cross–references, and produce various reports. It contains Troff macros for the following functions:

- declare a global prefix for a requirements document
- declare global keywords for the curtrent requirements document
- declare a document identifier
- start and end a requirement, identified by the identifier of the current document and a number unique in this document.
- declare a reference to a requirement identifier (in the current or in another document)
- add explanations to a requirement
- add keywords to a requirement
- assign a weight to a requirement.

In addition, there are macros that allow one to create:

- a requirement table,
- a list of implementation decisions,
- a requirement checklist,
- a systems engineering priority recommendation list.

The RADIX tool is used in combination with a method in which all engineers at different system aggregation levels are required to use this tool for documenting requirements. After a feature requirements list is generated, each feature receives an "owner" who will be the expert of this feature and will serve as a consultant for other engineers. The owner must be present during later reviews. Test engineers use the feature requirements to design tests; idem for performance verification engineers.

4.2.4 DOORS

DOORS is a commercially available hypertext system intended for use as traceability tool. It allows the definition of bidirectional links between points in different text documents. Attributes may be defined for the linked items (such as priority, source, version, date, author) as well as for the links (e.g. strength of the link, inconsistency or agreement of the linked items). Programs can be associated with the links, that are triggered when the links is traversed.

Regardless of the technology (database or cross-references), the current generation of commercially available traceability tools typically provides the following functionality:

- **Storage of links between items.** The items may be requirements, design items, explanations, etc. They may be represented as fixed format database records or free format text. Links may be annotated, e.g. with degree of strength.
- **Storage of links between texts**. The texts may be requirements documents, design documents, etc.

- Storage of requirements in free text format with a **hierarchical numbering scheme.**
- **Reporting facilities.** Examples are keyword searches, the traversal of links, producing cross-reference lists, producing traceability matrices, etc.

The tools have no model of system development and cannot distinguish the different kinds of links that can exist between items. Database tools and tools based on text processors typically provide static reporting functionality, whereas hypertext-based tools provide on-line browsing functionality.

4.3 Techniques to represent links

In this section, we look the ways used to represent links between items that must be traced.

4.3.1 Matrices

Description. A simple way to represent links between items is a matrix in which the horizontal and vertical dimension list the items that can be linked, and the entries in the matrix represent links between these items. The items in both dimensions may or may not be be the same.

Evaluation. The term "matrix" is a bit high–brow, for it is not a matrix in the mathematical sense. The matrix representation is equivalent to a graph representation, but it is less orderly as a visual presentation technique. Only finite lists can be represented, but this is not really a restriction. More importantly, only binary links between items can be represented. Some links may have a higher arity.

An advantage of the matrix representation is that it is easy to understand. It provides a format that can be discussed by stakeholders with different backgrounds.

Examples. The matrix representation is used in the ARTS system [9, 11] to link requirements with subsystems. It is used in the Quality Function Deployment (QFD) method to link user quality attributes with design requirements, or design requirements with design requirements (to indicate agreement or conflict) [5, 16, 38]. QFD is a method to translate user requirements into design requirements.

4.3.2 ER models

Description. Links between items can also be represented by Entity–relationship (ER) models. The linked items are entities, the links are relationship instances.

Evaluation. The ER representation has the advantage that links with arity higher than 2 can be represented. Moreover, an ER model of links can be implemented using any database technology. This view of has been taken by many repository designers.

The use of database technology has the advantage that ad hoc query and reporting facilities are easily available.

Examples. ER models of items and links are used in the specification of the repository of the ESE system [32] Marconi's Requirements and Traceability Management tool, RTM [17], by the Maestro II Software Engineering Platform [37], and by the Project Master Database for software engineering environments [29]. It is not clear from the published literature whether the the Automated Requirements Traceability System, ARTS [9, 11], uses an ER model.

4.3.3 Cross-references

Description. A requirements specification is a document with many cross-references among parts of the document, as well as with references across different documents. The relevant links are then embedded as pointers in a text, which may be an informal natural language text or a formal specification. Even links between diagrams can be viewed as cross-references.

Evaluation. The use of cross-references is simple to understand, and software that maintains cross-references and can produce reports about them, can be implemented easily. Existing packages like LaTeX already contain cross-referencing facilities.

Cross-referencing is useful for written specifications but not for a concise representation of links, such as can be done with matrices. Cross-references are always binary links, so that links of higher arity cannot be easily represented.

Examples. Low-tech representation of cross-references are the cross-reference system of LaTeX and the systems described by Jackson and Yu. In Jackson's [20] system, requirements are written in natural language documents, that are all available on-line and that contain cross–references defined by the author of the document. The cross–references are used to extract dependent items and to produce various reports. Yu [42] describes the RADIX system, a Troff macro package to format requirements, define cross–references, and produce various reports. Both systems have the important property that they are (claimed to be) used, and that the users (requirements writers) are (claimed to be) content.

Several researchers have constructed prototypes of hypertext systems, possibly extended with multimedia functions to present and animate requirements using sound or moving images [22, 14].

4.4 Discussion

Histories. None of the techniques explicitly involves the representation of histories, although some papers mention version dates. Database systems that support entity and relationship histories, or hypertext systems that support histories of texts and links, are not mentioned. Histories and versioning is routinely done in configuration management (CM), so the CM literature is a place to look for techniques. Another place to look is research into historical database systems (e.g. TSQL etc.). I am not aware of hypertext systems that support histories.

Most techniques allow the representation of attributes of links as well as of the linked items. Examples of such attributes are priority (of requirements) and strength (of links).

A set of items and links represented in a database can be queries by a query language. There is no query language for matrices and hypertext, so one often sees these systems supplemented by a query language. Because networks must be traversed, relational calculus is not powerful enough: one needs to compute the transitive closure of a reachability relation and for that one needs recursive queries. Thus, one needs the power of at least Datalog combined with the power of a historical query language (like TSQL).

Complex links. Edwards and Howell [10] propose complex links in which a link can be an and-or tree. The root and leaves of the tree are items. The root is linked by an and-or expression to the leaves. For example, a requirement can be satisfied by (design item 1 and design item 2) or (design item 2 and design item 3). A similar idea can also be found in Mylopoulos, Chung and Nixon [28].

They also propose representing the conditions under which the link exists (e.g. the conditions under which a design item satisfies a requirement), and storing an attribute that indicates whether the requirement is partially or completely satisfied by design elements. They note that nonfunctional

requirements tend to be satisfiable only by an entire system and this satisfaction cannot be traced to a particular set of design elements.

Chapter 5

The management of traceability

All of the research effort and most commercial products focus on tool support. Only the RADIX tool comes with some management advice, summarized in section 5.1. Quality Function Deployment (QFD) is a low-tech method to keep the focus of all stakeholders in development on user requirements (section 5.2). In section 5.3, problems reported with realizing traceability are summarized. Section 5.4 draws some conclusions.

5.1 RADIX

Yu [42] describes a method used in combination with the RADIX tool.

- Requirements are partitioned into a list of *feature requirements*. Each feature requirement is assigned to an engineer who plays the role of *requirements owner* throughout the development process. The owner is a consultant for the developers, is present at reviews such as design reviews and test plan reviews, and in general must ensure that the requirements that he or she owns are implemented completely and correctly.
- Feature requirements are allocated to one or more design units who must implement the feature. Backward and forward tracing are provided to ensure that all requirements are implemented completely and correctly.
- All links between feature requirements, architectural requirements and design requirements are listed by the RADIX tool before development starts. All engineers are responsible for keeping these links up to date.
- All tests are given a unique test identification number and are linked to requirements. The feature requirements owner uses this to ensure that his or her features are implemented completely and correctly.

5.2 Quality function deployment

QFD is a species of integrated product engineering [5, 16, 38]. In **integrated product engineering**, all people involved in the product life cycle are present in one development group: marketing, programmers, designers, analysts, maintenance engineers, finance, etc. First, user's or market desires are collected using marketing techniques such as focus groups. User requirements may be annotated with a priority. Next, the development group meets to link user requirements to design requirements

trough traceability matrices in which the strength of the links is entered in the cells of the matrix. By multiplying strength with priority and adding the results for design requirements, a priority for design requirements can be derived. Furthermore, design requirements are linked to each other, where the links are annotated to indicate agreement or conflict between requirements. Conflicting design requirements that jointly satisfy a high priority user requirement require technological research to reduce the conflict. The procedure can be repeated to link design requirements to implementation requirements, or to production and manufacturing requirements, etc.

The important point is that *all* people involved in the development process participate in one development group. Nobody is allowed to miss a meeting. Because participants range from marketing to product engineers, all become aware of the issues relevant to others: engineers become aware of the problems and objectives of marketing, and marketing become aware of the constraints and objectives of engineers. In addition, the group discussions have traceability matrices as their focus.

5.3 Problems with realizing traceability

The following problems have been reported with the implementation of traceability in the development process.

- Problems with allocation and flowdown:
  - Top level requirements are not created until the analysis phase is completed [11].
  - Requirements are not allocated to subsystems before they flow down [11].
  - There is no real flowdown of requirements: Requirements are just repeated at the next lower aggregation level.
  - Some requirements cannot be allocated to any particular design component (e.g. required programming language, nonfunctional requirements) [11].
- Management problems
  - The effort to maintain traceability is not perceived by developers and managers to be cost-effective [12]. It is considered by management to be extra, optional work, for which insufficient resources are allocated. This agrees with the opinion of engineers that maintaining traceability information is costing them too much work [11]. Capturing information on the design history of a project may take over 50% of an engineers time [36, page 24]. The amount of data to be maintained is considerable. From 1000 to over 20000 requirements may have to be managed [11, 36]. Despite all this work, the benefits of maintaining traceability are not clear: in projects where tools and techniques to maintain traceability are used, problems with traceability are still reported [12].
  - There is a prejudice that the management of the requirements is done by those not intelligent enough to do the requirements engineering [11].

The second problem may be the key to the explanation of the first: If traceability is maintained in a software developmement organization that has not yet reached the required maturity, this efort may actually be counterproductive. As in all cases of automation, the adage is: organization comes before automation. People working at the lowest maturity level are likely to look down upon any work that is not directly concerned with technical development.

5.4 Discussion

To repeat, the problems with realizing traceability may be related to the maturity level at which a development organization operates. At low maturity levels, the benefits of maintaining traceability information are not perceived. The major problems of realizing traceability are organizational, not technical. Note that a successful method like QFD is extremely low-tech (there are tools for manipulating the QFD matrices). If we view realizing traceability as the implementation of a tracking information system in the development organization, it becomes apparent that user procedures must be defined in addition to a data model. Worflow analysis and definition may be helpful here.

Chapter 6

Discussion and conclusions

Further research is needed in the following topics:

- **Models**
  - Comparing and integrating the models produced by Ramesh et al., Wieringa [41, chapter 13], [40] and the models discussed in chapter 3 to yield one integrated model of the development process. In MCM [40], the relationship between semi-formal (diagram-based) and informal representation techniques has been defined in detail and this relationship should be made part of the model.
  - Identifying an essential core model, to be used for a simple tracking system, and an elaborate model that extends the simple model.
  - A more theoretical research question is the representation of versions and history. There are several research prototypes of historical database systems, and this topic really lies outside the topic of traceability. Nevertheless, we could investigate what particular historical query needs arise in a tracking information system.
  - The addition of versions and histories to hypertext systems also needs investigation.
- **Software support**.
  - See if we can combine the advantages of database and data models on the one hand with those of hypertext (DOORS) and cross-references (LaTeX) on the other. Should we interface these different software tools? Can we use data models to add structure to a hypertext system?
  - Test the models in TCM.
- **Management**.
  - Study the configuration management [1, 2, 3, 6, 4, 18, 39] and change management [24, 25, 26] literature and see what we can learn from it regarding traceability management, version control tool support, etc.
  - Relate the models and tools to SEI maturity levels, using Humphrey [18] as a lead.
  - Investigate how the tracking information system can be used. The management problems listed above show that there is a problem here. One way to approach this problem is to use workflow definition techniques to define the management of links. Another way is to identify the maturity level of the development organization and provide traceability at that level and not higher. (These two approaches can be combined.)

For the 2RARE project, we should identify the maturity level of the software development process at Belgacom and identify the needs for tracking information in this process, assuming that the Albert language and the Oblog Case tool are used. Based on these needs, we should define a data model and usage procedures and implement a suitably simple tracking database that meets the identified needs. Since we are required to use DOORS, we must find a way to translate the data model into DOORS. We should investigate how DOORS can be linked to Oblog, how it can be used to store Albert specifications, and how it can be linked to a document processing system. DOORS' active links could be used to assist in impact analysis and in general to reason about the links. Experience with the tracking system should be monitored in order to improve the system as well as the software process.

Bibliography

[1] A. Alderson. Configuration management. In J.A. McDermid, editor, *Software Engineer's Reference Book*, pages 34/1–34/17. Butterworth/Heinemann, 1992.

[2] L.J. Arthur. *Software Evolution: The Software Maintenance Challenge*. Wiley, 1988.

[3] W.A. Babich. *Software Configuration Management, Coordination for team Production*. Addison-Wesley, 1986.

[4] E.H. Bersoff, V.D. Henderson, and S.G. Siegel. *Configuration Management: Investment in Product Integrity*. Prentice-Hall, 1980.

[5] P.G. Brown. QFD: echoing the voice of the customer. *AT&T Technical Journal*, pages 18–32, March/April 1991.

[6] W. Bryan, C. Chadbourne, and S. Siegel, editors. *IEEE Tutorial: Software Configuration Management*. IEEE Computer Society, 1980.

[7] A.M. Davis. *Software Requirements: Objects, Functions, States*. Prentice-Hall, 1993.

[8] M. Dorfman. System and software requirements engineering. In R. Thayer and M. Dorfman, editors, *System and Software Requirements Engineering*, pages 4–16. IEEE Computer Science Press, 1990.

[9] M. Dorfman and R.F. Flynn. ARTS — an automated requirements traceability system. *The Journal of Systems and Software*, 4:63–74, 1984.

[10] M. Edwards and S.L. Howell. A methodology for systems requirements specification and traceability for large real-time complex systems. Technical Report NAVSWC TR 91-584, Naval Surface Warfare Center (Code U33), 10901 New Hampshire Avenue, Silver Spring, Maryland 20903-5000, U.S.A., September 1991.

[11] R.F. Flynn and M. Dorfman. The automated requirements traceability system (ARTS): an experience of eight years. In R. Thayer and M. Dorfman, editors, *System and Software Requirements Engineering*, pages 423–438. IEEE Computer Science Press, 1990.

[12] O.C.Z. Gotel and A.C.W. Finkelstein. An analysis of the requirements traceability problem. In *IEEE International Conference on Requirements Engineering*. Computer Science Press, April 1994.

[13] O.C.Z. Gotel and A.C.W. Finkelstein. Contribution structures. Technical report, Imperial College, 1995.

[14] P.A. Gough, F.T. Fodemski, S.A. Higgins, and S.J. Ray. Scenarios — an industrial case study and hypermedia enhancements. Technical report, Philips Research Laboratories, Cross Oak Lane, Redhill, Surrey RH1 5HA U.K., 1995.

[15] G.A. Harrington and K.M. Rondeau. An investigation of requirements traceability to support system development. Technical report, Naval Postgraduate School, Monterey, CA 93943, 1993.

[16] J.R. Hauser and D. Clausing. The house of quality. *Harvard Business review*, 66(3):63–73, May–June 1988.

[17] S.G. Howard. Requirements and traceability management. In *Colloquium on Tools and Techniques for Maintaining Traceability During Design*, Savoy Place, London WC2R OBL, U.K., 2 december 1991.

[18] W.S. Humphrey. *Managing the Software Process.* Addison-Wesley, 1989.

[19] IEEE. Ieee guide to software requirements specification. In M. Dorfman and R.H. Thayer, editors, *Standards, Guidelines, and Examples on System and Software Requirements Engineering*, pages 16–38. Computer Science Press, 1990.

[20] J. Jackson. A keyphrase based traceability scheme. In *Colloquium on Tools and Techniques for Maintaining Traceability During Design*, Savoy Place, London WC2R OBL, U.K., 2 december 1991.

[21] M. Jarke, K. Pohl, C. Rolland, and J.-R. Schmitt. Experience–based method evaluation and improvement: a process modeling approach. In *IFIP WG8.1 Conference CRIS'94*. 1994.

[22] H. Kaindl. The missing link in requirements engineering. *Software Engineering Notes*, 18(2):30–39, 1993.

[23] R.C. Laubengayer and J.S. Spearman. A model of pre–requirements specification (pre–RS) traceability in the department of defense. Technical report, Naval Postgraduate School, Monterey, CA 93943, 1994.

[24] N.H. Madhavji. The Prism model of changes — part 1: Introduction. Technical Report TR-SOCS-90.15, School of Computer Science, McGill University, July 1990.

[25] N.H. Madhavji. The Prism model of changes — part 2: Dependency structure. Technical Report TR-SOCS-90.16, School of Computer Science, McGill University, July 1990.

[26] N.H. Madhavji. The Prism model of changes — part 3: Change structure. Technical Report TR-SOCS-90.17, School of Computer Science, McGill University, July 1990.

[27] N.H. Madhavji. Environment evolution: the Prism model of changes. *IEEE Transactions on Software Engineering*, 18(5):380–392, 1992.

[28] J. Mylopoulos, L. Chung, and B. Nixon. Representing and using nonfunctional requirements: a process-oriented approach. *IEEE Transactions on Software Engineering*, 18(6):483–497, June 1992.

[29] M.H. Penedo and E.D. Stuckle. PMDB — a project master database for software engineering environments. In *9th International Conference on Software Engineering*, pages 150–157, 1985.

[30] K. Pohl, R. Doemges, and M. Jarke. PRO–ART: PROcess based Approach to Requirements Traceability. Technical report, Informatik V, RWTH–Aachen, Ahornstraß 55, 52056 Aachen, Germany, 1994.

[31] C. Potts and G. Bruns. Recording the reasons for design decisions. In *10th International Conference on Software Engineering*, pages 418–427, 1988.

[32] C.V. Ramamoorthy, Y. Usuda, A. Prakash, and W.T. Tsai. The evolution support environment system. *IEEE Transactions on Software Engineering*, 16(11):1225–1234, November 1990.

[33] B. Ramesh, A.G. Abbott, M.R. Busch, and M. Edwards. An initial model of requirements traceability: An empirical study. Technical Report NPS-AS-92-025, Naval Surface Warfare Center Dahlgren Division, 10901 New Hampshire Avenue, Silver Spring, Maryland 20903-5000, U.S.A., September 1992.

[34] B. Ramesh and M. Edwards. Issues in the development of a requirements traceability model. In S. Fickas and A. Finkelstein, editors, *International Symposium on Requirements Engineering*, pages 256–259. IEEE Computer Science Press, 1993.

[35] B. Ramesh, G. Harrington, K. Rondeau, and M. Edwards. A model of requirements traceability to support systems development. Technical Report NPS-SM-93-017, Naval Surface Warfare Center Dahlgren Division, 10901 New Hampshire Avenue, Silver Spring, Maryland 20903-5000, U.S.A., September 1993.

[36] B. Ramesh, T. Powers, C. Stubbs, and M. Edwards. A study of current practices of requirements traceability in systems development. Technical Report NPS-SM-93-018, Naval Surface Warfare Center Dahlgren Division, 10901 New Hampshire Avenue, Silver Spring, Maryland 20903-5000, U.S.A., September 1993.

[37] Softlab. *Maestro II Software Engineering Platform, Release 3.1*, 1994.

[38] M. West. Quality function deployment. In *Colloquium on Tools and Techniques for Maintaining Traceability During Design*, Savoy Place, London WC2R OBL, U.K., 2 december 1991.

[39] D. Whitgift. *Methods and Tools for Software Configuration Management.* Wiley, 1991.

[40] R.J. Wieringa. A method for building and evaluating formal specifications of object-oriented conceptual models of database systems (`MCM`). Technical Report IR-340, Faculty of Mathematics and Computer Science, Vrije Universiteit, December 1993.

[41] R.J. Wieringa. *Requirements Engineering: Frameworks for Understanding.* Wiley, 1996. To be published.

[42] W.D. Yu. Verifying software requirements: a requirements tracing methodology and tool — RADIX. *IEEE Journal on Selected Areas in Communication*, 12(2):234–240, 1994.