Chapter 5

REQUIREMENTS TRACEABILITY

Francisco A. C. Pinheiro
Universidade de Brasília

Abstract: Requirements tracing is inevitable. We do tracing when we search information and it is difficult to imagine a software development environment without some tracing aids. For medium to complex systems we should have a traceability model and traceability aids should be in place. In these systems we have a quite complex web of relationships. It is common to have several requirements coming from the same source as well as a single requirement having more than one source. It is also common to have one requirement deriving several others as well as several requirements collapsing into a single one. The diversity and huge amount of information dealt with when developing large software systems point to the need for automated support to development practices, including traceability. This chapter presents the concept of requirements tracing and discusses several aspects related to traceability. Particular importance is given to the informal aspects of requirements tracing and to the non-functional nature of requirements traceability.

Key words: Requirements traceability, Requirements management, Requirements change, Functional traces, Non-functional traces.

1. THE CONCEPT OF TRACING

The concept of requirements tracing is quite simple: to follow relationships or links. It is essential for software development because a lot of information is used and produced and it should be kept related (traceable). When developing large systems it is hard to remember all links that were made to relate information and it may even be impossible, in a distributed or multi-team development, to know that such links exist. Moreover, the matter is complicated when immediate links to a given piece of information are not what is required. One usually wants to follow a chain of links that goes beyond the starting point of tracing.
In ordinary usage a trace is defined as a mark, track, sign, etc. showing what has existed or happened [27]. The mark does not need to be a material thing. It can be, for example, a non-material indication or evidence of the presence or existence of something, or of a former event or condition [27]. The existence of a former event or condition is stressed in the usual definitions of requirements traceability. The IEEE standard 830-1984 [19] states that:

*A software requirements specification is traceable if (i) the origin of each of its requirements is clear and if (ii) it facilitates the referencing of each requirement in future development or enhancement documentation.*

In the natural world the tracing of something amounts to look for, or to follow the course, development, or history of traces, or occurrences of traces left by whatever is being traced. Requirements traceability is the ability to trace requirements, and to this end more than a traceable specification is necessary. Gotel and Finkelstein [15] express the concept in a more complete way:

*Requirements traceability refers to the ability to describe and follow the life of a requirement, in both a forwards and backwards direction (i.e., from its origins, through its development and specification, to its subsequent deployment and use, and through all periods of on-going refinement and iteration in any of these phases.)*

Gotel and Finkelstein's definition has become the common definition of requirements traceability. They make explicit that in order to follow (i.e., to trace) the life of a requirement you have to describe it. Two important aspects of requirements tracing are used to extend their definition. The first one is the ability to capture the traces we want to follow, and the other is the idea that traces should be viewed as natural occurrences, naturally produced.

A requirement is traced back to a document because the requirement is supposed to carry traces of the document. It may be that the requirement is directly extracted from the document, it may be that the document contains statements supporting the requirement, or it may be something else. In any case, the existence of some influence of one on another is necessary if they are to be viewed as related objects. In the same way, a requirement is traced forward to a design module because the module is supposed to carry traces of the requirement. Again, it may be that the module is designed to meet the requirement, it may be that the module specifies a test procedure for the requirement, or it may be something else.

The following working definition acknowledges the natural occurrences of traces and the fact that we have to identify which traces should be controlled, how they may be captured, and how they may be followed afterwards [31]:
Requirements traceability refers to the ability to define, capture, and follow the traces left by requirements on other elements of the software development environment and the traces left by those elements on requirements.

It should be clear that the software development environment involves not only the technical, but also the social aspects of software development. Its elements comprise not only the technical artefacts such as specifications, diagrams, and code, but also people, policies, decisions, and even less tangible things like goals and concepts.

2. TRACEABILITY MODES

There are several ways in which requirements tracing can be performed. As regards the direction of tracing, a requirement may be traced in a forward or backward direction; as regards requirements evolution, a requirement may be traced to aspects occurring before or after its inclusion in the requirements specification; and as regards the type of the objects involved, we may have inter or extra-requirements traceability.

2.1 Backward and forward traceability

The concepts of forward and backward traceability, illustrated in Figure 5-1, have standard definitions in the literature [44]:

Forward traceability is the ability to trace a requirement to components of a design or implementation.

Backward traceability is the ability to trace a requirement to its source, i.e. to a person, institution, law, argument, etc.

*Figure 5-1. Backward and forward traceability*
A requirement is traced forward, for example, when the requirement is changed and we want to investigate the impact of the change. We may want to get all related test procedures to decide which tests should be added, modified, or removed to make the test procedures comply with the changed requirement. We may also want to get the components built to meet the requirement and analyse the changes which should be made in those components.

A requirement is traced backward, for example, when there is a change and we want to understand it, investigating the information used to elicit the changed requirement. We may want to know who the person interested in the requirement is, or what the documents from which the requirement was extracted are, or to which departments in the organisation the requirement is related.

2.2 Pre and post-requirements specification traceability

A requirements specification is the result of the elicitation process. The tracing of a requirement can be done in either way: to get information related to the process of elicitation, prior to its inclusion in the requirements specification, or to get information related to its use, after the requirement has been elicited and included in the requirements specification. Figure 5-2 illustrates the concept of pre-requirements specification (pre-RS) traceability and post-requirements specification (post-RS) traceability [15].

**Pre-RS traceability** refers to those aspects of a requirement's life prior to its inclusion in the requirement specification.

**Post-RS traceability** refers to those aspects of a requirement's life that result from inclusion in the requirement specification.

![Figure 5-2. Pre-RS and post-RS traceability](image)

Pre-RS traceability is useful, for example, when there is a change to a requirement and we want to get the requirement's sources or the people supporting it to validate the change. There is usually an interest in investigating the reasons why a
requirement was conceived in the way it was. If we know that the person who approved a requirement was in charge of the sales department we may want to know about other requirements approved by the same person, or other requirements coming from the same department; or we may want to know about other departments closely interacting with the sales department.

Post-RS traceability is useful, for example, to get the design module to which a requirement was allocated or the test procedures created to verify the requirement.

## 2.3 Inter and extra-requirements traceability

A great deal of work is devoted to build a precise requirements specification, with the right amount of details, appropriate for the intended readers. In doing that, requirements are commonly rephrased and refined, new requirements appear, derived from the old ones, and some are abandoned. The ability to trace the links between requirements is called inter-requirements traceability. The links between requirements and other artefacts are captured by extra-requirements traceability.

**Inter-requirements traceability** refers to the relationships between requirements.

**Extra-requirements traceability** refers to the relationships between requirements and other artefacts.

Inter-requirements traceability is important for requirements analysis and to deal with requirements change and evolution. We do inter-requirements tracing, for example, when we want to know all requirements derived from a given requirement, or its chain of refinement.

## 2.4 The scope of traceability modes

The definitions given to the traceability modes should be relaxed to more closely reflect the ways we do tracing. For example, the backward and forward traceability were defined as starting in a requirement and going backward or forward to other objects. This is inappropriate since in many occasions the starting point of tracing is not a requirement. If we get an unexpected result in a test procedure it may be necessary to trace the test procedure back to its related requirements in order to understand the results. The same happens if we have a change in some organisational procedure or policy: we may want to get all requirements already elicited from these sources to see if they remain valid.

As illustrated in Figure 5-3, the scope of traceability modes overlap. Backward and forward tracing may happen at the same time as pre-RS tracing or post-RS tracing. When tracing a requirement forward to the code built to meet the requirement we do post-RS and forward tracing. In the same way, we do post-RS
and backward tracing when we investigate an error, tracing a component back to the
requirement it was intended to satisfy. We may even think of inter-requirements traceability and pre-RS traceability at the same time. Just imagine an iterative
development with new requirements appearing in each iteration. We do pre-RS and
inter-requirements tracing when we trace these new requirements, which are not yet
part of any specification, to investigate their relationships with the already specified
requirements from previous iterations.

3. FUNCTIONAL AND NON-FUNCTIONAL TRACING

Non-functional traces are those traces related to intentions, purpose, goals,
responsibilities, and other intangible concepts. On the other hand, the functional
traces are those related to well established mappings between objects.

3.1 Functional tracing

Functional tracing is related to the functional aspects of software development,
those that may be described in terms of transformations on well-defined states of
software development, its models, and its artefacts.

Functional traces occur naturally when we use well-defined models and
notations to describe objects. The traces are derived from the syntactic and semantic
connections prescribed by the models or notations used. An example is the
relationships between diagrams, classes, attributes and methods in object-oriented models.

Given a model we can always functionally trace an object represented in that model by following its syntactically defined relationships. If the tracing depends on the semantics of the model, then it will be functional only if the semantics is precisely defined. For example, the relationship between a requirement and a test case may be syntactically defined. Nevertheless, establishing a link between them is no guarantee that the test case is indeed a test for the requirement, unless the semantics of the relationship is such that it allows a test case to be related to a requirement only if it tests the requirement.

To functionally trace an object to another there should be a mapping between the two objects. This mapping is expressed with respect to the models characterising the objects being traced and their relationships. Some model types and the mappings they allow are:

a) Analysis models, relating objects used in the requirements phase of the development, e.g., interviews and their transcriptions, documents and the requirements extracted from them.

b) Design models, relating objects used in the design phase, e.g., classes and attributes, sequence diagrams and methods. These mappings tend to be more structured.

c) Process models, relating objects of the development process, e.g., tools and activities, artefacts and people. These models may include management issues.

d) Organisational models, relating organisational objects, e.g., organisational structures and people, goals and people, activities and resources. These models may include environment and social issues.

There may be functional traces between objects described by different models of the same type. This is the case of relationships between a process in a data flow diagram and the classes whose objects are responsible for implementing (part of) that process. It is also the case of relationships between an entity in an entity-relationship diagram and a repository in a data flow diagram. Functional traces may also exist across models of different types. If there is a mapping relating requirements (in an analysis model) to processes (in a design model) designed to satisfy them, then we can functionally trace a requirement and get its associated processes. Traceability inside a model is sometimes called \textit{vertical traceability} and between different models is sometimes called \textit{horizontal traceability} [24].

The models can overlap. For example, some elements of an organisational model may be present in a process model. Finding common elements is a good starting point to devise mappings relating different models. Also, the models do not need to be explicitly stated. For most of the informal activities of software development there are implicit models with well established and uniformly understood
relationships between objects, e.g., a meeting and its meeting minute, a manual and a computer system, a policy and a memorandum documenting it.

Functional tracing properly answers questions like "What are the test cases assigned to this requirement." In these cases it is sufficient to trace the requirement following its syntactically defined relations to test cases. On the other hand, if we are trying to answer the question "What are the tests which test this requirement," this can be done in a functional way only if the semantics of the relationship is sufficient to guarantee that every test associated to a requirement is indeed a test for the requirement. Otherwise, we are dealing with non-functional traces and we have to resort to interpretation and analysis of the contents of the related objects.

3.2 Non-functional tracing

Non-functional tracing is related to the tracing of non-functional aspects of software development. They are usually related to quality aspects and result from relationships to non-tangible concepts.

The traces that relate requirements to goals, objectives, intentions, and decisions are examples of non-functional traces. Most traces involving non-functional requirements are also non-functional traces. It is very difficult to decide which component carries traces of a non-functional requirement. For a performance requirement, for example, this amounts to ask which component is responsible for achieving the required level of performance — there may be no definite answer in terms of single components.

The aspects covered by non-functional tracing are classified in four groups:

a) Reason, comprising, among others, the traces related to the justification and interpretation of requirements and other artefacts, to the definition of goals and purpose, and to the explanation of activities.

b) Context, comprising, among others, the traces related to the environment of both the application domain and the software development process, including organisation and social aspects;

c) Decision, comprising, among others, the traces related to the decisions made during the software development process, and to the communications among the stakeholders and developers.

d) Technical, comprising, among others, the traces related to non-functional requirements.

These groups may overlap and in practice non-functional traces are seldom classified into only one group. For example, a reason is usually situated in a context and it may be given to support some decision related to a non-functional requirement.
5. Requirements Traceability

Non-functional traces are not amenable to direct reference. This is much like non-functional requirements that need to be translated to functional correspondents in order to be verified. A non-functional trace has to be re-expressed in terms of functional ones.

The tracing of a non-functional aspect of software development can be automatically performed only using a representation of that aspect. Therefore, we have to use some model to functionally capture the non-functional aspects we want to trace, e.g., we may use an organisational model to relate policies, goals and roles to requirements, or we may use a process model to relate requirements to activities and resources.

It is important to remember that when using models much of the non-functionality may be lost. Suppose a requirement is traced back to a discussion and, as a result, we obtain a report of the discussion and information about the time when it happened and who was there. This may be of great help but surely the rich web of personal interactions is lost. The development of more complex models to accurately register the non-functional traces may not be the answer. The production of traces and their capture are as important as the traces themselves, and a model may be just too complex to be efficiently used.

3.3 Traceability usage

People do not trace for the sake of tracing. Tracing is an auxiliary activity to achieve some goal and the view that reduces tracing to the retrieval of pre-defined relationships is very bureaucratic and at most serves as a starting point for the purposeful action of looking for related information. Tracing serves both management and technical people. In both cases it is a very dynamic activity, guided by necessity and not by any pre-defined structure.

An example of a technical objective is the tracing to investigate the causes of a test error. Suppose an engineer who has found an error during a verification procedure. He decides to get all the modules being tested by the test suite. He browses each one of them and decides to go further back to the design documents and requirements related to those modules. Having seen the requirements, he can go forward and look at the testing modules to see some detail and back again to the requirements being tested. If needed, he can retrieve information about the people responsible for the faulty requirements. This activity may proceed going back and forth until he is satisfied or while he perceives he is making some progress.

An example of a management objective is the tracing to decide upon a change request. A manager decides to get all requirements related to the proposed change and proceed forward to get the modules implementing them. Then he traces each module back to get any additional requirement related to the module. Having the set of modules impacted by the change, he may get the programmers responsible for them, their actual assignments and their schedules. Again, he might continue
obtaining information in this way until he is satisfied and while he feels he is making progress.

Ramesh [34] identifies two kinds of traceability users: the low-end users and the high-end users. Low-end users are mostly involved with technical issues. Therefore, their needs are related to solving technical problems and making technical decisions. Typical needs of low-end users would be to know if all requirements derived from a given requirement have been agreed upon, or to get all requirements allocated to a given component. High-end users are more interested in managerial issues. Typical needs of high-end users would be to know what information should be captured, by whom, and how it should be used. They would also require that the critical elements of the software process should be traced to their stakeholders and they would be interested in traces regarding process-related information and costs.

Non-functional tracing is necessary to satisfy the needs of the high-end users of traceability: their needs are mostly of a non-functional nature and more difficult to be fulfilled in an automated way. Differently, the needs of the low-end users of traceability are mostly satisfied with functional tracing.

4. FORMAL AND INFORMAL NEEDS FOR TRACING

When dealing with large amounts of information, and for complex systems, it is essential to have some form of automated aid for tracing. A traceability environment, composed of procedures, methods, techniques, and tools, should be in place to help the process of tracing. Usually the environment is built around a model that gives a uniform view for the traces and traceable objects.

4.1 The need for formality

The use of formal traceability models helps in automating the tracing process. Not only is the automation of the model facilitated, but also the automatic generation of traces and the definition of procedures to verify consistency and correctness of traces. Using a formal traceability model, it is easier to enforce the appropriate registration and extraction of traces and traceable objects according to the interpretation of appropriateness supported by the model.

The benefits of formality do not come only from formal descriptions of traceability models. Functional tracing is made more precise and its scope is enlarged when some other elements of the software development process are formally described. For example, the use of a formal specification language to specify the system being developed makes possible the definition of traces involving not only references to the specification as a whole, but also references to its constituent parts such as sorts, functions, operators, and data types.
5. Requirements Traceability

The use of formality increases traceability efficiency. It provides a more precise way of discerning which traces to follow if the need for tracing is identified with objects that have been formally described. If there is a formal mapping between two formally described objects, then the relationships among them can be automatically derived, enhancing the functional aspects of traceability.

4.2 The informal nature of software development activities

Informality is needed to deal with the fundamentally unstructured way in which information is gathered and used [12, 30]. Information itself is in many cases better viewed as unstructured. Therefore, what should be made traceable is in many cases inherently informal, e.g., natural language statements, interview's transcripts, and images.

The reduction of the informally expressed information to some formal structure may facilitate its manipulation but it is likely not to fulfil traceability needs. Most of the time what relates a text, a discussion, or an image to a requirement is not their formal features, if there is any. Instead, they are related based on many different grounds such as relevance, proximity, causality, policy and even organisational politics. The reason to relate objects is in itself an important information to be made traceable. The rationale behind each decision is usually what is sought when tracing software objects.

The informal and unstructured aspects of information are proper of non-functional traces. We may use models to partially capture some of the non-functional aspects of software development. For example, we may use rationale systems [26] to capture design decisions; we may use a model like IBIS [6] to capture the discussion process in terms of issues, positions, and arguments; or we may use a model like i* [46] to capture the goals and intentions arising in a particular organisational context. However, not all aspects of non-functional traces can be captured in a functional way. The use of models is important but serves only as a starting point to guide the process of tracing non-functional traces.

4.3 Traceability environment circularity

Most traceability environments impose a sort of circularity in the process of tracing: we only trace what has been registered, and we register only when we perceive a trace. This is illustrated in Figure 5-4. To understand how this happens, consider the following steps performed to set up and use a traceability environment:

1. First, at the definition phase, we build a traceability model defining the potential traces.
2. Second, at the production phase, we perceive the actual traces, at the moment or soon after they occur, and register them into the environment.
3. Third, we have an actual need for tracing, a need to seek information related to some object.
4. Forth, at the extraction phase, we trace the desired information using the retrieval mechanisms of the environment to follow the already registered traces.

Observe that at step 2 we are not fulfilling any real tracing need, but just following a procedure to make the perceived traces available for future use. Nevertheless, the traces have to be identified or perceived in order for them to be registered and this amounts to a sort of tracing beforehand.

Suppose that a document contains evidence supporting a requirement but, for some reason, people do not perceive that connection and fail to register the trace between the document and the requirement. In this situation the trace still exists, because the document still contains the supporting evidence, but it will be unavailable from inside the traceability environment.

The need to trace beforehand is present in almost all traceability techniques: requirements are tagged with information known to be relevant; cross references are made with objects known to be related; and traceability matrices are filled with known connections. The current support for traceability is restricted, to a great extent, to the extraction of information already made traceable.

Even if the traceability environment allows the automatic registration of traces, there still may be some circularity in the process of tracing. A trace is automatically registered according to some definition of traces. Therefore, a trace can be followed only if its type has been previously defined in the model.

Some traceability models are based on the definition of very rich sets of traces and traceable objects, in an attempt to devise all conceivable (or at least reasonable) traces. This is not a complete solution because the complexity of the model may impair the tracing process.

![Figure 5-4: Tracing circularity](image)

One way to avoid the circularity problem is to allow the recreation of the situations in which the traces occurred, e.g., by means of hyper-media objects as
described by Haumer et al. [17]. It is also useful to incorporate extraction mechanisms not bounded to the existence of explicit links. Two works on the capture of implicit traces are El-Ramly et al. [10], proposing a method to trace user interface interactions using knowledge discovery and pattern searching algorithms, and Antoniol et al. [3], describing the use of information retrieval techniques to recover links between code and texts documents. Automatic generation of traces is described in [48, 40, 42].

5. MODELLING TRACES

A traceability model is essential to automate any part of the tracing process. The model is a central component of a traceability environment around which the tracing procedures, methods, and tools are organised.

Three aspects should be covered by a traceability model: the definition, the production, and the extraction of traces. The definition is related to the specification of the traces and traceable objects; the production is related to the capture of traces, usually by means of an explicit registration of the objects and their relationships; and the extraction is related to the actual process of tracing, i.e., the retrieval of registered traces.

5.1 Trace definition

A traceability model should define its trace units, i.e., how the traceable objects are represented in the model. It should also define its traces and which sorts of traces may exist, indicating what the trace is intended to represent, what the trace units that may be involved are, and under what conditions traces may be registered.

Figure 5-5 shows the main elements of an abstract traceability model. It is the underlining model of TOOR [31], a formal object-oriented traceability environment. In TOOR's model a trace is represented by a binary relationship between two traceable objects, its source and target objects. One may follow a trace forward, going from a source to a target object, or backward, going in the opposite direction.

![Figure 5-5. TOOR's trace units](image-url)
The TOOR's model is abstract and its classes should be subclassed to reflect particular tracing needs. This is done by means of a project specification defining the traces and trace units for each project. A meta model for traceability is presented by Ramesh and Jarke [36].

5.1.1 Informal needs for trace definition

The definition of traces and traceable objects should promote their uniform understanding. Differences in interpretation are the cause of errors, and in the more serious cases one may end up tracing what did not happen. Suppose that someone registers a relationship supports between document $D$ and requirement $R$, meaning that the document contains evidences of the requirement. Later on, someone else wants to get all requirements extracted from document $D$ and decides to follow the supports relationship (because the relation is misunderstood). The requirement $R$ will be retrieved and it may not be true that the requirement was indeed extracted from the document.

To account for non-functional traces, the definition of the traceable objects should allow the use of hyper-media objects like videos, recordings, and images, together with mechanisms for inspecting these kinds of objects. The relationship between recorded real world observations and parts of conceptual models is called extended traceability in [17].

5.2 Trace production

The trace production is the actual occurrence of traces. It roughly corresponds to the pragmatics of a traceability model and it should be taken into account if a traceability model is to be useful. The production of traces encompasses their perception, registration, and maintenance. The registration of traces is called "installation of traces" in [8].

It is not enough to devise potential traces and to define the ways in which they may be registered. A trace should be perceived when it occurs; otherwise it will go unnoticed and will not be registered.

The description of traces by a traceability model should resemble the ways traces occur in the real world. If there is a mismatch between what is specified in a traceability model and what may be actually captured, we may end up first registering what is not there and second retrieving what never happened.

There are several ways in which trace production may be impaired by a misconceived traceability model. For example, it would not make sense if in a model we had a requirement related to a diagram under a responsibility relationship because we do not usually expect a document to be responsible for a requirement. Also, relating a requirement to a description of its economical impact would be practical only in very restricted environments. In those situations where economical
impacts are dealt with in broad terms and not in a requirement by requirement basis, it would be unwise to ask developers to think of, and register, the economical impact for every requirement.

5.2.1 Informal needs for trace production

Trace production is an important aspect of traceability models not only because one can trace only what is available, but also because it may interfere directly with the activities of developing software. It may impose an overload on people carrying out these activities [14]. The less intrusive the capture of traces, the more efficient and accurate the use of the traceability model is.

Trace production also deals with the maintenance of already registered traces. Changes to requirements and other objects may result in new traces being produced and may also cause modifications in the already registered traces. Cleland-Huang et al. [5] present an event-based traceability model to cope with changes in traceable objects.

For functional traces, the trace production itself can sometimes be automated. Domges and Pohl [7] describe the use of trace strategies to guide the tracing process. Zisman et al [48] and Spanoudakis [40] present a formal approach that supports the automatic generation and maintenance of traceability relations between requirements artefacts and object models. The dynamic identification of architectural information based on pre-defined types of traceability links is described in [42]. For non-functional traces, by definition, it is not possible to have a completely automated capture of traces. In the case of non-functional traces a traceability model might be more useful if it allows the registration of richer objects such as videos, recordings, and images. The use of real world scenes for tracing purpose is described by Haumer et al. [17].

5.3 Trace extraction

A traceability model should provide different and flexible ways to retrieve the information registered in it, so that the most appropriate one can be chosen for each occasion. The following modes of tracing should have appropriate aids in terms of trace extraction mechanisms [31]:

Selective tracing to restrict the tracing to certain selected patterns of objects and relations. One could trace considering only certain classes of objects and relationships. In more advanced models the tracing could be restricted to some notion of context, a development phase for example.

Interactive tracing to allow interactive browsing over a set of related objects with each step being guided by the possible relationships. One could trace
forward or backward starting from an object, and then trying different alternative paths following the object's relationships. There should be features to help in the navigation process.

Non-guided tracing to allow a user to go from one object to another at will, inspecting contents as desired. This is convenient when you have little information on what or how to trace.

5.3.1 Informal needs for trace extraction

The tracing mechanisms and the way the traces and the related objects are visualised should be thought as an aid for the process of tracing. It should effectively help to fulfil the need that triggered the tracing. The extraction procedure and the results of tracing should be intuitive; otherwise it may be regarded as not very useful in some circumstances. If we only get a list of related objects and no justification for the relationships, or if the trace extraction mechanism is not intuitive enough, it may be simpler to go around and to informally ask people if they know about any relevant information.

6. TRACEABILITY SUPPORT

This section presents a brief description of the more relevant approaches to support requirements traceability by means of models, methods, techniques, and processes. A similar list may be found in [44]. Commercial tools are not addressed here. Andrile [2] contains a tools database including descriptions of commercial tools.

6.1 Traceability models

Traceability models provide a representation for traces and trace units. They establish the structures containing the elements and the relations used in tracing, usually specifying their types as well as the constraints under which the elements of the model can be related.

The REMAP model [35] enlarges the Issue-Position-Argument model of IBIS [6] and allows the capture of design rationale by providing support for discussion and negotiation of design issues.

Contribution Structures is a model that addresses personnel-based traceability [13, 14], making traceable the human sources of requirements, requirements related information, and requirements related work. It consists of a web of relations among contributors and the artefacts resulting from their contributions.
Document centred models usually represent traces as relations between documents of different types. An example is SODOS [18] that also allows the tracing of software life-cycle aspects. Other examples are hypertext models like RETH [21, 8] and HYDRA [32]. Hypertext models are particularly suitable for capturing informal or originally unstructured information. Using hypermedia features they allow the assessment of information in its original format, being suitable for non-functional tracing. The use of web technologies to aid the tracing of requirements is described by Song et al. [39].

Database guided models are used to register trace information on databases for later retrieval. Toranzo and Castro [41] present a comprehensive database model as a metamodelling tool for traceability.

Design-centred models are based on artefacts used in the design process. Letelier [23] describes a framework based on a UML meta model that allows the construction of traceability models using the extension mechanisms of UML. The use of UML for requirements tracing is also discussed by Tsumaki and Morisawa [43]. Eyged [9] uses scenario information to generate traceability links between elements of design models. Wieringa [45] shows how documents and design artefacts can be coherently linked, and Bianchi et al. [4] describe a case study analysing several design-centred traceability models.

6.2 Traceability methods

Traceability methods define an organised set of activities and establish the procedures necessary to make artefacts related to each other according to some model. The model need not be explicit. Instead the procedures may be centred around some general concept of related elements and make use of traceability techniques.

The RADIX method [47] is built around a document based traceability model. Trace units are marked parts of texts inside documents and the method itself consists of several steps to organise the production, use, and verification of these documents.

6.3 Traceability techniques

Traceability matrices are one of the best known traceability techniques. The requirements are associated to the lines and the other objects of software development, like programs and test procedures are associated to the columns of a matrix. The matrix cells are marked to show the relationship between a requirement and an object. It is easy to devise more sophisticated ways of using traceability matrices, e.g., using different marks to indicate different kinds of relationships.

Cross references and indexing schemes are usually implemented as references made across several artefacts, to indicate links between them; or as lists of indices containing the related artefacts for each entry. Cross references are used, for
example, as part of the RADIX method [47] and indexing is implemented in READS [38].

Traceability techniques may easily be incorporated into several methods and may also be used in conjunction with different models. Indeed, almost all traceability tools implement some form of traceability matrix.

6.4 Tracing languages

There are a number of specification languages with features allowing references to requirements. For example, the statements traces-to and traces-from in the requirements specification language RSL [1] and the statement by-requirements in the prototype system description language PSDL [25] are used for tracing purpose. However, they are not languages primarily intended to requirements tracing.

Most of the current requirements tracing tools use conventional database query languages to inspect and retrieve trace information from their databases. Regular expressions are used as part of the tracing model implemented by TOOR [31].

6.5 Traceability processes

Many of the traceability information related to functional traces may be captured as a by-product of the development process. Software engineering tools are particularly suitable for this purpose, naturally integrating and maintaining information related to the activities and processes they support. A good and well-documented example of the automatic production of traceability information as a result of development practices is given by Pohl and Jacobs [33].

Configuration management is a good example of process related traceability because it is highly dependent on traceability mechanisms. It keeps different versions, sometimes representing different stages of evolution, of the same system and allows the development team to deal with the whole system in a consistent way with all components belonging to the same configuration. Configuration management has been very successful because the traceability mechanisms are in place to accomplish a very specific objective: to keep together those components that make up a configuration. The general questions to be answered are "if I change this component, which others should be changed?" or "if I get that version of this component, which others should I also get to keep a consistent set of components?"

Configuration management traceability is essentially functional. Usually the components making up a configuration — specifications, source and executable code — have precisely defined relationships. Of course, the concept of configuration can be extended to encompass other artefacts like requirements, modules, tools, and even people; but these cases are harder to work with because the relationships
assume a non-functional nature making traces more difficult to be perceived and registered.

The requirements baseline model proposed by Leite [22] uses configuration management concepts to control the evolution of requirements and other artefacts of the software development process.

6.6 Tracing libraries

A tracing library may implement a broad range of components, from techniques like traceability matrices to whole models. They are usually based on abstract models and are used to implement concrete models. The proposal in Pinheiro [29] is based on the TOOR's abstract model. It is intended to implement the necessary structures to maintain a configuration state of objects and relations and to develop mechanisms to update and retrieve information from it. Gao et al. [11] present a set of guidelines to build traceable components and describe a Java framework addressing component traceability and maintenance issues. Tracing libraries seem to be a promising way to facilitate the incorporation of traceability features into software development environments and applications, and the adaptation of traceability models to different needs.

6.7 Reference models

Reference models are used to organise concepts in a coherent framework. They serve as a basis for the construction of particular models that adhere to the characteristics prescribed by the reference models. Therefore, a reference model saves time and effort when used to guide the construction of particular solutions. Moreover, different solutions based on the same reference model may be compared and, sometimes, used interchangeably.

The traceability reference model proposed by Ramesh and Jarke [36] is built around a simple meta model, based on three basic elements:

a) the **stakeholder** representing people who have an interest on requirements and on the tracing of requirements,

b) the **source** representing the origins of a requirement and the artefacts used for documentation purposes, and

c) the **object** being traced.

There are also basic traceability types representing the relationships between the various model elements. A knowledge management system based on this traceability reference model is described in [26].

A more general reference model for requirements and specifications is proposed by Gunter et al. [16]. Although their reference model is not intended specifically for
traceability purposes, it is a good starting point to discuss traceability issues and to devise useful traceability mechanisms.

7. CONCLUSIONS

The adoption of requirements traceability practices do not produce uniform results [28]. There are several difficulties related to the implementation and use of traceability. Wieringa [44] points out that the major problems of realising traceability are organisational, not technical. Jarke [20] identifies the adaptability to project specific needs as a critical issue and Dömges and Pohl [7] cite integration into the process, adaptation to the situation, and support for organisational knowledge creation as desirable features of traceability environments. Rose [37] argues that it is important to have information on the assignment of tasks to individuals and organisational hints to specific expertise.

The informal aspects of tracing and the non-functional nature of some traces explain most of the difficulties of the tracing process. The discussion in this chapter may be used to draw some practical implications summarised in the following guidelines:

As regards trace definition a traceability model should:

1. Define few basic traces.
2. Allow the specification of user-definable traces.
3. Allow the use of rich representations of traceable objects.

As regards trace production:

4. A traceability environment should use as much automation as possible to capture traces.
5. When not done automatically, traces should be identified by people who are able to understand what they are and how they may be used in the future.
6. Traces should be registered as close as possible to the point where they are produced.

As regards trace extraction:

7. A traceability model should provide different and flexible ways for extracting the information registered in it.
8. The extraction of traces should be context sensitive.
9. The extraction procedure should, as much as possible, give the information needed to satisfy the tracing needs, and not just links to where the information may be.
The most important implication drawn from this chapter is that we should not strive to fully automate the tracing process. Instead, we should devise traceability environments with a right mix of automated and non-automated aids, capable of dealing with the functional and non-functional tracing, and with an adequate support to all aspects of tracing: the definition, the production and capture, and the extraction of traces. The focus being in solving problems — the problems for which we need to trace information.

ACKNOWLEDGMENTS

The author would like to thank Jorge Doorn and Julio Cesar Leite for their help in improving the contents and presentation of this chapter.

REFERENCES


5. Requirements Traceability


[33] Pohl K. and Jacobs S. Traceability between cross-functional-teams. Proceedings of 1st International Conference on Concurrent Engineering, Research and Application; 1994; August, Pittsburgh, USA.


[40] Spanoudakis G. Plausible and adaptive requirement traceability structures. Proceedings of SEKE’02, 14th International Conference on Software Engineering and Knowledge Engineering; 2002 July 15-19; Ischia, Italy.


