Softw Syst Model (2010) 9:529–565
DOI 10.1007/s10270-009-0145-0

THEME SECTION

A survey of traceability in requirements engineering and model-driven development

Stefan Winkler · Jens von Pilgrim

Received: 15 January 2009 / Revised: 24 November 2009 / Accepted: 1 December 2009 / Published online: 22 December 2009

Abstract Traceability—the ability to follow the life of software artifacts—is a topic of great interest to software developers in general, and to requirements engineers and model-driven developers in particular. This article aims to bring those stakeholders together by providing an overview of the current state of traceability research and practice in both areas. As part of an extensive literature survey, we identify commonalities and differences in these areas and uncover several unresolved challenges which affect both domains. A good common foundation for further advances regarding these challenges appears to be a combination of the formal basis and the automated recording opportunities of MDD on the one hand, and the more holistic view of traceability in the requirements engineering domain on the other hand.

Keywords Requirements engineering · Model-driven engineering · Model-driven development · Traceability

1 Introduction

Traceability—the ability to describe and follow the life of software artifacts [105]—has been used as a quality attribute for software for several decades now. Actively supporting traceability in a software development project can help ensuring other qualities of the software, such as adequacy and understandability. On the other hand, neglecting traceability can lead to less maintainable software and to defects due to inconsistencies or omissions.

The largest part of traceability research so far has been done in the last two decades by the requirements engineering community. Over the past years, it has gained in importance, and traceability topics have become subject to research in many other areas of software development. One of these areas is model-driven development[1] (MDD), an area where parts of the software development process are executed automatically using model transformations. Since traceability is mainly achieved by documenting different aspects of (usually manual) transformations of software development artifacts, MDD seems to be able to leverage traceability by automatically generating these documentations.

However, traceability practices in general are far from mature, benefits are to a large part not conceived in the industry, and we are still standing at the beginning of an emerging discipline. A lot of research—both fundamental and applied—has still to be done. This is a challenge, not only because of the difficult research questions, but also because researchers in the field of traceability are usually part of very different larger research communities (such as requirements engineering, modeling, or program understanding), and there is only little communication between these communities. Nevertheless, communication between traceability researchers is essential, as several topics (e.g., link types and unifying traceability scheme—see Sect. 2.3) are often discussed independently in these communities. This bears the risks of not

Communicated by Prof. Richard Paige.

S. Winkler (✉) · J. von Pilgrim
FernUniversität in Hagen, 58084 Hagen, Germany
e-mail: stefan.winkler-et@fernuni-hagen.de

J. von Pilgrim
e-mail: jens.vonPilgrim@fernuni-hagen.de

[1] There are several terms of the form *model-driven something*. In our impression most of these terms refer to the same thing, namely the application of models, modeling activities, and model transformation to software development. For simplicity, we only use the term MDD in this article.

reaching a common understanding and of reinventing the wheel.

Therefore, we want to pull together the fields of requirements engineering and MDD with this survey and present both an overview and a comparison of the state of research and practice in both areas. Our goals for this survey were to address both new and experienced researchers. We want to encourage the reader to stand back and take a more global view on traceability: There are multiple aspects which are relevant not only in a specific domain and should, therefore, be discussed across the boundaries of those domains. In addition, particularly in the field of requirements traceability, there is no recent survey of the literature which would provide an easy and up to date introduction into that area. Hence, we want to take the opportunity to give an updated view on the state of research. We build on earlier literature surveys by Aizenbud-Reshef et al. [1], Dahlstedt and Persson [44], Galvão and Göknil [67], Palmer [142], Pinheiro [145], Spanoudakis and Zisman [171], von Knethen and Paech [193], Walderhaug et al. [195], and Wieringa [200] and complement them with our findings of a systematic analysis of recent publications. To achieve this, we have investigated the main conferences, workshops, and journals of both the RE and MDD communities for the recent years and identified those contributions which refer to traceability. From that starting point and from the earlier literature surveys mentioned before, we have followed citations and references using web-based literature search engines. Finally, we have tried to build the body of knowledge which forms the core of this article by classifying the results into the categories

Basics Basic description of traceability and associated topics.

Working with traces Descriptions of how traceability can be achieved and used.

Practice Description of the state of the practice and what limits the application of traceability in the industry.

Solutions Description of approaches to overcome these limitations.

Within these basic categories we tried to find suitable subcategories (see Table 1).

Naturally, due to the fact that traceability is rooted in requirements engineering (see Sect. 2.1) and because traceability in RE has a research history of more than two decades now, there are substantially more publications in the requirements area. Nonetheless, we have tried to separate between requirements traceability and traceability related to models. As it turned out, however, regarding *solutions* a clear separation between these topics is not possible, as there are no sharp borders, and there are both model-related approaches in requirements traceability research and requirements-oriented approaches in MDD.

Table 1 Outline of the main topics

Section	Topic
2	Basic principles of traceability
2.2	Terminology (RE in Sect. 2.2.2, MDD in Sect. 2.2.3)
2.3, 2.4	Formalization (RE in Sect. 2.3.1, MDD in Sect. 2.3.2)
3	Working with traces
3.2	Usage (RE in Sect. 3.2.1, MDD in Sect. 3.2.2)
3.3, 3.4	Visualization and usability
4	Limitations of traceability in practice
5	Research: overcoming limitations (See overview in Table 2)
6, 7	Future challenges and conclusion

Eventually, we have derived the organization of this article which is listed by topic in Table 1 for convenience. We start in Sect. 2 by describing the current basic knowledge about the field of traceability: We discuss the roots and history of the topic, terminology, schemes, metamodels, and traceability link types. Next, in Sect. 3 we show the basic activities related to traceability and in particular we focus on how traces can be used to achieve certain goals in a development project. Since there are some differences in how some of these topics are conceived in RE and MDD, respectively, we have separated our analysis into specific subsections where appropriate.

After this more theoretical groundwork, we look at the state of the practice in Sect. 4 and we derive factors which hinder the adoption of traceability practices in the industry. This is followed by a thorough classification and description of recent research in Sect. 5 and a discussion of open issues in Sect. 6. In the conclusion in Sect. 7, finally, we summarize the differences and commonalities of requirements traceability and traceability in MDD and conclude this article.

2 Basic principles of traceability

Traceability is still an emerging discipline and several terms and definitions are not entirely clear or agreed upon. Therefore, the following sections define the terminology used in this article. At the same time traceability research has its roots in several different domains, which results in quite different understandings of the area and terminology. In order to understand these different viewpoints, we first give an overview of the different research areas in which traceability is rooted.

2.1 Roots of traceability research

It is difficult to answer the question where traceability research is located exactly in the software research community

as it overlaps with different domains of interest. Historically, traceability has started as an area of requirements engineering, which is in turn a part of software and systems engineering. This is indicated by the fact that most definitions of traceability explicitly refer to requirements traceability. But nowadays, traceability is also seen as a method to manage traces of artifacts other than requirements, or as an instrument to generally follow the whole software development process.

In requirements and software engineering, traces are particularly valuable in the area of validation and verification where they help to identify pairs of relating artifacts which can then be validated and verified against each other, and in software maintenance, which is, for example, concerned with program understanding.

Traceability is also a topic in the field of model-driven software development, where traces can be recorded automatically as a by-product of model transformations. Traces recorded that way can be used to keep the models consistent and to support change propagation between models which are derived from each other.

In areas other than software engineering, traceability research overlaps with knowledge engineering, project and process management, and others. Although this description of the areas in which traceability is addressed is not complete, it makes clear that traceability is a very heterogeneous area of research. As a consequence, different people assign very different goals to traceability research and perceive desirable granularity and quality very differently depending on their point of view.

In this article, we will concentrate on the fields of requirements engineering as the field in which most research on traceability has been done, and MDD, a field in which traces play an important role as an integral part of the result of model transformations.

2.2 Terminology

2.2.1 General terms

Generally, a software development process involves the creation of unique intermediate and final products. The classical development process has as input a single high-level goal, or product vision. This goal is then gradually transformed into requirements with different levels of granularity. In turn, these requirements are reworked into architecture, design, test cases, documentation, and code. Each of these products is verified and validated against its sources, and if the final acceptance test passes, the software is deployed. Nowadays, the transformations from the vision to the product usually happens both incrementally and iteratively, but the sequence of transformational steps from an initial idea to the final system is basically still the same for most software development processes.

In this context, Traceability is defined in the IEEE Standard Glossary of Software Engineering Terminology [90] as

1. The degree to which a relationship can be established between two or more products of the development process, especially products having a predecessor–successor or master–subordinate relationship to one another. [...]
2. The degree to which each element in a software development product establishes its reason for existing.

These "elements in a software development product"—such as documents, models, or code—are referred to as *software configuration items* by Pressman [151] or as *artifacts*, for example, by Gotel [73] and Ramesh and Jarke [156]. In MDD, only models are used, or in case of model-to-text transformations, text.[2] In this article, we will use the terms *artifact*, or *model* in the context of MDD. In addition, artifacts can be recursively composed from other artifacts in accordance with Derniame et al. [50]. For example, a *software requirements specification document* could include a *business object model diagram* as a figure, which in turn could be further decomposed into single *business objects* and so on.

Throughout the software development process, artifacts are created and modified either because new requirements or insights have emerged, or because an artifact is transformed into another, more concrete one as a product of a development activity. This results in *traces* left on the artifacts. The IEEE Standard Glossary [90] simply defines a trace as "a relationship between two or more products of the development process." According to the OED, however, a *trace* is defined more generally as a "(*possibly*) *non-material indication or evidence showing what has existed or happened*" [166]. If a developer works on an artifact, he leaves traces: The software configuration management system records who has worked on the artifact, when that person has worked on it, and some systems also record which parts of the artifacts have been changed. But beyond this basic information, the changes themselves also reflect the developer's thoughts and ideas, the thoughts and ideas of other stakeholders he may have talked to, information contained in other artifacts, and the transformation process that produced the artifact out of these inputs. These influences can also be considered as traces, even though they are usually not recorded by software configuration management systems.

[2] Often, model-to-text transformations generate code, and we subsume code under text as long as it is not parsed. In this latter case, an abstract syntax tree is created which can be interpreted as a model.

Consequently, according to Pinheiro [145], traces can be divided into two different categories:

Functional traces These are created by transforming one artifact into another using a defined rule set. The transformation is not required to be performed automatically, but it has to follow unambiguous procedures, such as transcribing an audio recording of an interview. Jacobson [93] and Lindvall [111] attribute this as a *seamless* transformation. A subset of functional traces is the set of *explicit traces* which are either created together with the artifact as a by-product of the transformation, or which can be unambiguously reconstructed at any time by analyzing the original artifact, the transformed artifact, and the transformation rules. Functional traces, and particularly explicit traces are also usually found in models with formally defined syntax and semantics.

Non-functional traces Pinheiro categorizes them further into the aspects *reason*, *context*, *decision*, and *technical*. They refer to traces of informal nature. These traces result from more or less creative process, such as semantically analyzing and extracting customer requirements from a set of meeting minutes. In the technical category, non-functional traces could, for example, exist on the parts of the code which are affected by quality requirements.

Note that the IEEE Glossary lists a second definition of *trace*: "A record of the execution of a computer program, showing the sequence of instructions executed, the names and values of variables, or both." This definition is usually referred to as an *execution trace* and applied in the field of dynamic program analysis. To avoid ambiguities, the term *trace* refers to the descriptions given above. The latter concept will be explicitly referred to as *execution trace*.

2.2.2 Requirements traceability

In the domain of requirements engineering, the term *traceability* is usually defined as the ability to follow the traces (or, in short, to *trace*) to and from requirements. Two common definitions of requirements traceability are given by Pinheiro [145] as

> 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.

and by Gotel and Finkelstein [74] as

> 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 periods of on-going refinement and iteration in any of these phases).

Usually, these definitions are also implicitly (e.g., in [13, 111]) or explicitly [105] extended to general traceability of all artifacts as the ability to define, describe, capture, and follow traces from and to artifacts throughout the whole software development process, which seems sensible because all artifacts of a software development are (or at least should be) driven by requirements.

From the description of traces in the previous section, it can be observed that a trace can in part be documented as a set of metadata of an artifact (such as creation and modification dates, creator, modifier, and version history), and in part as relationships documenting the influence of a set of stakeholders and artifacts on an artifact. Particularly those relationships are a vital concept of traceability, and they are often referred to as *traceability links*. Traceability links document the various dependencies, influences, causalities, etc. that exist between the artifacts. A traceability link can be unidirectional (such as DEPENDS- ON) or bidirectional (such as ALTERNATIVE- FOR). The direction of a link, however, only serves as an indication of order in time or causality. It does not constrain its (technical) navigability—traceability links can always be followed in both directions.

Over the years, several other terms related to requirements traceability have been established. The most common ones are *pre-RS*, *post-RS*, *forwards*, *backwards*, *horizontal*, and *vertical traceability*. These are shown in Fig. 1 and described in the following.

Gotel and Finkelstein [74] have introduced and emphasized the classification in *pre-requirements specification* (*pre-RS*) *traceability* and *post-requirements specification* (*post-RS*) *traceability*. Pre-RS traceability is concerned with traces occurring during elicitation, discussion, and agreement of requirements until they are included in the requirements specification document. This includes dealing with informal, conflicting, or overlapping information. Post-RS traceability is concerned with the stepwise implementation of the requirements in the design and coding phases. It includes documenting the traces of the diverse manual and automatic transformation steps eventually producing the system. Gotel and Finkelstein argue that post-RS traceability is more trivial than pre-RS traceability, because it only has to uncover the traces of these transformation steps, which are comparatively formal. Because pre-RS traceability has to cope with communication problems and informality, it is the more challenging of the two [74].

The ANSI/IEEE Std 830–1984 [89] has introduced the terms *backward traceability* and *forward traceability*. Backward traceability refers to the ability to follow the traceability links from a specific artifact back to its sources from which it

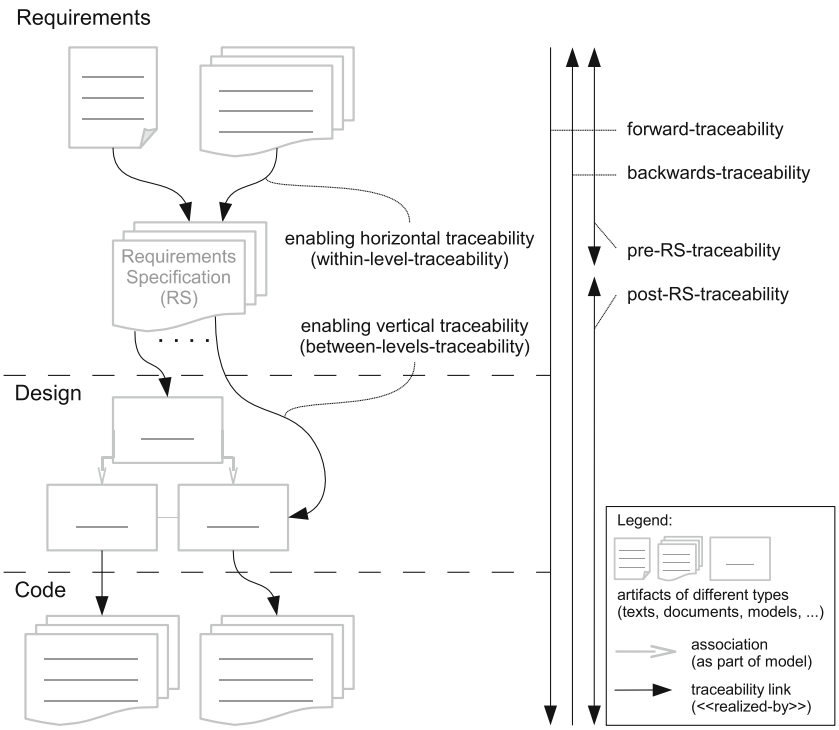

Fig. 1 Dimensions and directions of traceability links

has been derived. Forward traceability stands for following the traceability links to the artifacts that have been derived from the artifact under consideration.

Finally, Ramesh and Edwards [155] have introduced the distinction between *horizontal* and *vertical* traceability. These terms differentiate between traceability links of artifacts belonging to the same project phase or level of abstraction, and links between artifacts belonging to different ones. Because levels of abstractions or phases can both be drawn from left to right or from top to bottom, there is some uncertainty about which term is used for which dimension. The case more commonly reported is illustrated in the example in Fig. 1: A traceability link between a statement documented by meeting notes and a requirement in the requirements specification document is considered a part of horizontal traceability. A link from a requirement to a design element, like a class or a component, is considered a part of vertical traceability. It is, however, unclear if traceability practices really differ depending on the different types of links and, hence, if a classification in horizontal and vertical traceability is actually sensible in this context.

Yet, in the context of iterative development processes which include the two dimensions *increment* and *iteration* the difference does make sense: Antoniol et al. [12] briefly relates horizontal and vertical traceability to the iteration- and increment-dimensions of the development process, respectively. This differentiation has an immediate effect on the types of traceability links, as horizontal traceability mostly implies a form of EVOLVES- TO link.

2.2.3 Traceability in MDD

In the context of MDD, traces partially fulfill the same purpose as in requirements engineering because in many tasks, MDD is simply an automation of software engineering. The special characteristic of MDD is the usage of models and automated transformations. So, the artifacts under study are mainly (intermediate) models. This context influences the definitions and semantics of the terms known from requirements traceability and software engineering in general.

In addition, the "MDD way" to define terms is often to simply define models and metamodels in which they occur. This is why most publications either do not refer to an explicit definition of traceability at all, or only refer to the general IEEE definition cited above. Also, since *traceability* cannot be modeled intuitively, most definitions refer to *traceability links*.[3]

An example for a model-like definition for the term traceability is the rather technical and narrow definition that is given by the OMG [132]:

> A trace [...] records a link between a group of objects from the input models and a group of objects in the output models. This link is associated with an element

[3] It can be noted that the distinction between traceability (the ability) and traceability links (the relations) is not regarded very strictly by some authors.

> from the model transformation specification that relates the groups concerned.

The weakness of this definition is that it limits the meaning of a traceability link in MDD to that of a by-product of a model transformation. As modeling and MDD, however, do not exist in isolation but are used as part of a development process including manual tasks and even tasks going beyond the scope of modeling, a broader understanding of traceability is needed.

Two definitions of traceability from the MDD research community will be repeated here. The one by Paige et al. [141] describes traceability as

> [...] the ability to chronologically interrelate uniquely identifiable entities in a way that matters. [...] [It] refers to the capability for tracing artifacts along a set of chained [manual or automated] operations.

This definition is based both on the IEEE definition and the OMG's perspective on traceability, but is extended to include manual transformation steps. Still, traceability links according to this definition are limited to a sequential chain of transformation steps.

In contrast to this, Aizenbud-Reshef et al. [1] extend Gotel's definition of (requirements) traceability links to

> any relationship that exists between artifacts involved in the software-engineering life cycle. This definition includes, but is not limited to the following:
>
> – Explicit links or mappings that are generated as a result of transformations, both forward (e.g., code generation) and backward (e.g., reverse engineering)
> – Links that are computed based on existing information (e.g., code dependency analysis)
> – Statistically inferred links, which are links that are computed based on history provided by change management systems on items that were changed together as a result of one change request.

This definition regards both the case of links which are generated as a by-product of a transformation, as well as links which are generated by other algorithms. Yet, compared to the previous definition, they do not mention automatic versus manual creation explicitly.

In addition to those different definitions, there is also no commonly agreed basic classification. Yet, we can draw parallels to requirements traceability: Paige et al. [141] describe MDD as a development process which is started with non-model artifacts such as informal, natural language descriptions of requirements, spreadsheets, etc. Then, these artifacts are expressed as models which are gradually transformed. At the end of the model transformation process, code is generated. Similar to pre-RS- and post-RS-traceability, we could classify traceability from an MDD perspective as pre-model-, intra-model-, and post-model-traceability, denoting traceability between early artifacts and the first model, traceability between the gradually refined models, and traceability between the final model and the non-model artifacts generated or derived from it, respectively. Also, model transformation specifications themselves are artifacts and thus, need to be traceable, e.g., back to their requirements in order to leverage their reuse.

Another basic distinction of traceability links can be found in explicit vs. implicit traceability links. Yet, there is no common consensus: Paige et al. [141] describes explicit traceability links as links which are captured directly in models themselves using a suitable concrete syntax, and implicit traceability links as links which are generated or captured due to the application of model management operations. Simultaneously, Philippow and Riebisch [115] describe explicit traceability links as links which are explicitly expressed as connections, while implicit traceability links exist, e.g., between two elements in different models if they are identified by the same name, but not explicitly connected otherwise.

All in all, it can be seen that definitions and basic classifications of traceability in the MDD research community are still an area of ongoing discussions, and it will most certainly take some time until a consensus will be established.

2.2.4 Terminology used in this article

In this article, we differentiate between three essential terms, which we describe briefly in the following:

By *traceability* we mean the ability to describe and follow the life of a software artifact in the sense of the generalized definition originally presented by Gotel and Finkelstein [74]. We use this term also in word combinations to indicate that something is applied in the domain of research or achievement of traceability (e.g., traceability tools, techniques, practices, and so on).

A *trace* is a piece of (implicit or explicit) information which is an indication or evidence showing what has existed or happened [166]. We use this term to generally denote information that makes an artifact traceable. We do not imply a particular form or representation of this information.

Finally, a *traceability link* is, as already stated, a relation that is used to interrelate artifacts (e.g., by causality, content, etc.). Following our notion of a *trace*, a *traceability link* is a more concrete (but not the only) form of information that can be used to describe and follow certain aspects of the life of the respective software artifacts. Again, for this article, we do not imply a particular representation, syntax (binary, *n*-ary), semantic (type), or way of creation (manual recording, automatic derivation).

2.3 Traceability schemes and metamodels

The mere definition of the terminology of and around traceability helps in getting a basic picture. Yet, to really know what is a trace, and what its syntax and semantics are, templates and classifications have to be defined. In the field of requirements traceability, these are often called traceability schemes. A traceability scheme determines, for which artifacts and up to which level of detail traces can be recorded.

Several researchers, most of them engaged in the area of requirements traceability, have tried to analyze and define what would make a trace complete. The vision behind this work is to find a universal set of standards and instructions that determine the amount, semantic expressiveness, and other qualities of traces that should be recorded.

A traceability scheme thus defines the constraints needed to guide the (mostly manual) recording of traces. In most cases, if the traceability scheme is represented in a tool, it is technically implemented as a model or metamodel even in the requirements traceability area. The following sections describe the different views on traceability schemes and metamodels in requirements traceability and MDD.

The basic difference between both worlds is that in the former, traces are usually recorded manually and there is little formal guidance on how to perform this task, while in the latter, traces are usually an automatically recorded by-product of a transformation process. So, while requirements traceability has to deal with the question of "How much investment in traceability is just enough for this particular project?" questions in MDD are more specific and often more technical (e.g., "What could be a standard of traceability links which can be created as part of this transformation tool?" and even "How can we avoid automatically producing overwhelming amounts of irrelevant traceability links?" [195]).

At the same time, both worlds are continuously melting into each other, as nowadays there are a lot of approaches which try to express requirements as models, or at least in a model-oriented context. This development is twofold: On the one hand, there are *semantic models* such as goal models [185] or business object models which are used to express and relate domain concepts and which can—at least partially—replace natural language descriptions. On the other hand, there are *structural models* such as the requirements model in SysML [135] in which natural language still has an important role and in which model elements are mostly used to enable requirements to be addressed in a model-oriented way [53]. These alternatives and their consequences have to be addressed in future considerations.

2.3.1 Traceability schemes for requirements traceability

As described above, a central problem of requirements traceability is the economics behind applying traceability practices. This is why that research community tries to get insight in what could be achievable in theory, but also tries to find clear definitions which can be used to install and guide those traceability practices which are "just right" for a particular project.

If the economical factor is ignored, it is clear that best traceability could be achieved by recording any trace to the greatest possible extent. Thus, it is a research goal to define how much traceability could be achieved in theory, so that the results to that question can be clipped in order to eventually fulfill economic constraints. In order to gain insight on the topic of completeness, a research group around Ramesh [153,154,156,157] has observed traceability in several industrial projects over several years. Among other things, they have defined six core questions about artifacts that can be answered by traces:

What	information is recorded in the artifact? Is it a requirement, an assumption, an environmental constraint, etc. and on which other information is it based?
Who	has created or updated the artifact and documented the information? Which other stakeholders have been involved in that process and who belongs to the group of potential users of the information?
Where	has the information come from? What is the source of the information?
How	is the information represented? Is it documented as formal or informal text, or as graphics, or is it documented using audio or video recordings?
When	has the artifact been created, modified, or evolved?
Why	has the artifact been created, modified, or evolved? What is the rationale behind the task? Were there several alternatives? And why was this alternative chosen?

It seems clear that some of the questions are relatively easy to answer (such as *who* and *when*, if that information is kept in a controlled repository), while some questions are only possible to answer just after an artifact has been created or changed, and only by the stakeholders that have been involved. But if this information only remains in the minds of people and is not documented, it will get lost sooner or later. Finally, there is tacit knowledge (such as *why*), which is difficult to capture and to document. A traceability scheme helps in this process of recording traces and making them persistent.

One of the first, very basic conceptual models of traceability has been derived by Ramesh and Jarke [156] and is shown in Fig. 2. It contains three elements relevant to traceability:

Stakeholders	represent people involved in the software development process.

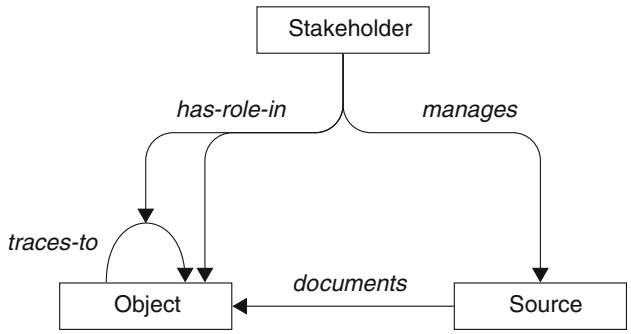

Fig. 2 The traceability conceptual model according to Ramesh and Jarke [156]

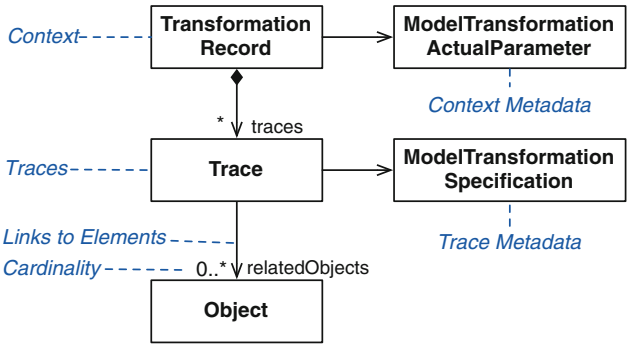

Fig. 3 The basic traceability metamodel

Sources stand for intangible or substantial sources of information including, for example, undocumented knowledge, policies, telephone calls, but also existing documents, standards, and laws.

Objects correspond to the traceable artifacts of the software, such as a single requirement or design element.

This model is certainly not a formal metamodel in the sense of MDD, but it was used in the past to explore the different forms and types of traceability.

While different communities and sub-communities concentrate on different aspects such as stakeholder identification and relation [6,73,140], or rationale modeling [29,42,56,57,122,150], most research concentrates on the nature of the TRACES- TO relationships or, as they are commonly called, traceability links.

Yet, a metamodel for traces itself is not enough. In order to systematically approach traceability, a schema specification is needed. Espinoza et al. [63] state, that such a schema specification has to include: a *traceability type set* which defines the types of links and their meaning for a project (see Sect. 2.4), a *traceability role set* which defines the stakeholders and their permission to access traces, a *minimal links set* that determines which links have to exist so that traceability information is regarded as correct and complete for a specific project. And finally, a schema specification must include a *metrics set* which defines quality measures for traceability in the respective project.

2.3.2 Traceability metamodels in MDD

In the context of MDD, traceability schemes are usually explicitly expressed in metamodels, which are also usually linked to models specifying model transformations. Currently there is no single standardized traceability metamodel.

The OMG has published a "proposal for an MDA foundation model" [132], in which the MDA—OMG's view of MDD—is defined and modeled. This proposal also includes a definition and model of a *transformation record*, which is a set of traces produced by a model transformation. The core of this definition is depicted in Fig. 3: An *n*-ary LINK relates model-elements (OBJECTS) together as the result of a model transformation specification, or transformational rule. This core model can also be found with minor variations in several research publications.

While the core conception of a traceability link is usually the same, most authors come up with ad-hoc traceability metamodels (e.g., [8,43,98,137,141,169,187,190,194]), extending the basic traceability model with their own conceptual or implementation-specific enhancements. As indicated by the italic labels in Fig. 3, we can identify several features found in most traceability models.

Figure 4 illustrates the common features of these metamodels we have extracted from the models found in literature. The feature diagram resembles the "traceability metamodeling language" (TML)—a language which can be used to design traceability meta-models—proposed by Drivalos et al. [54]. So in many cases it is possible to define a model with TML which reflects a possible configuration in the feature diagram and vice versa. However, while the feature diagram's main purpose is to illustrate the different characteristics of trace models to the reader, TML's focus is to technically enable model transformations between different trace models. The feature diagram is described below in more detail.

Usually, a context or container is used to store certain metadata, such as which metamodels were used or the values of the transformation parameters, if an automatic transformation was used to create the traces. Furthermore, the context serves as a container for the traces themselves.

Like the context, the trace can contain metadata, e.g., which rule or operation created the trace. There are two different kinds of models, one using untyped traces, others creating a sophisticated type hierarchy of traces. Of course, the type of a trace can be seen as metadata, and as such the type of a trace is sometimes just stored in a field. Additionally, traces

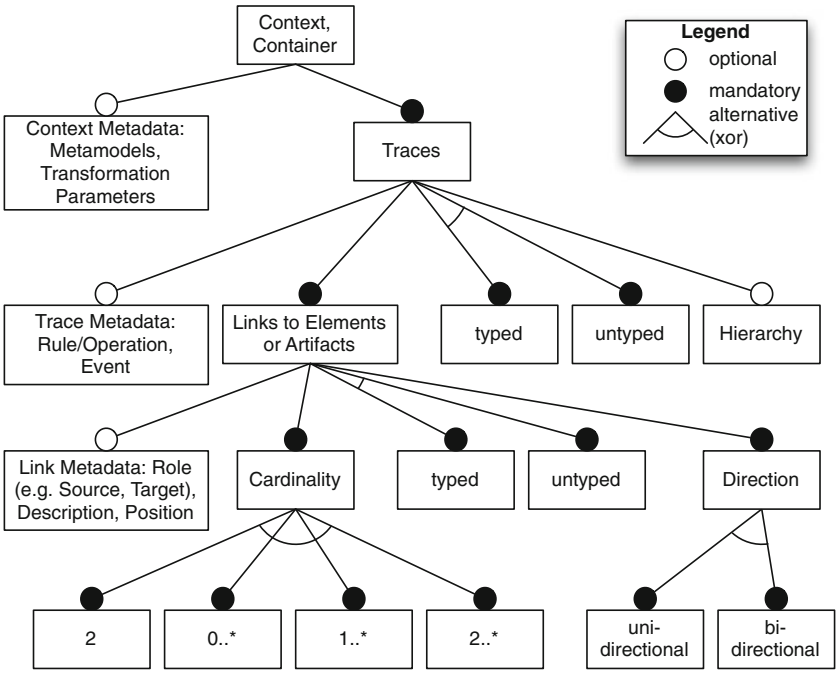

Fig. 4 Feature diagram of traceability models

can be composed recursively: a trace can contain sub-traces or dependent traces [8,70].

The most important feature of a trace is, of course, the actual links to the model elements or artifacts, which are connected via the trace. Again, every link can store metadata. In addition, roles are usually defined—in most cases as "source" and "target." Finally, in the model described by Oldevik and Neple [137], traces connect models with generated text and the links also store the position of the target element.

The cardinality of a link also varies in different models: The most restrictive variant is a single trace connecting exactly one source with one target element [8,190], which results in more traces. In others, more than two elements can be interconnected, enabling *m*:*n* relationships. Most models expect at least one source and one target element (2..∗), others allow pseudo traces with only one linked element (e.g., [141]) or even no elements (e.g., [132]).

Trace models are usually stored as separate models, and links to the elements are (technically) unidirectional in order to keep the connected models or artifacts independent. Alternatively, models can contain the links themselves or define the links as bidirectional. While this pollutes the models, navigation is much easier. As a compromise, Kolovos et al. [102] suggest model merging, which keeps the base models independent, but it is still possible to weave the links into the models if needed. Since the trace model is in most cases independent of the models it connects, the element links are usually untyped. In some cases, however, the trace model is aware of the connected models and the links may become typed.

2.4 Traceability link types

In addition to the aforementioned models, Dahlstedt and Persson [44], Espinoza et al. [63], Limón and Garbajosa [109], Ramesh and Jarke [156], Spanoudakis and Zisman [171], von Knethen and Paech [193], and many others have published suggestions for classifying traceability links based on their structural and semantic properties. These proposed classifications are difficult to evaluate and compare because there is no common level of abstraction and usually no unambiguous or formal definition of the different categories. Additionally, the categories themselves cannot be separated clearly: It is hard to tell, if a REFINES link (as in REQUIREMENT- REFINES- REQUIREMENT) belongs to one of the classes of *evolutionary*, *containment*, (*within-level*) *refinement*, or *dependency* links mentioned by von Knethen and Paech [193]. Depending on the viewpoint, it shows characteristics of all four of the classes. So, the semantics of a link is currently mostly dependent on the view and the interpretation of the stakeholders [158].

One of the more recent extensive classifications in the RE community has been presented by Spanoudakis and Zisman [171]. It has been derived from a thorough analysis of literature and defines eight categories of links:

Dependency links between two artifacts e_1 and e_2 describe that e_2 relies on the existence of e_1, or that changes in e_1 have to result in potential changes in e_2.

Refinement links are used in abstraction hierarchies to describe how complex artifacts are broken down into smaller, more concrete, or more manageable ones.

Evolution links are used when one artifact replaces another, like if a newer version replaces an older one, or if a set of overlapping requirements is reworked into a more elaborate set of requirements.

Satisfiability links denote another type of evolution during development: They are used to express that a downstream artifact is created complying with an upstream artifact, like a design element which satisfies one or more requirements.

Overlap links can be used when two artifacts describe common features or aspects of the system. As an example consider the representation of a requirement in natural language and in a formal notation.

Conflict links are used to express an existing conflict between two artifacts. In order to be useful, these links have to include information on how to solve the conflict, and on which alternatives are possible under which assumptions.

Rationalization links are also used to carry information about decisions, where they are used to document rationale behind the creation and evolution of artifacts.

Contribution links describe the various relations between stakeholders and artifacts.

Spanoudakis and Zisman [171] also provide a matrix containing pairs of artifacts and traceability link classes. This matrix gives an overview on which traceability links can connect which artifacts according to the literature. A similar list has also been created by Espinoza et al. [63].

In contrast to this flat classification, link types are also often organized in a hierarchical structure. The SysML specification [135], for instance, describes its different link types—DERIVE, VERIFY, COPY, SATISFY, and (implicitly) REFINE—as refined dependencies. The most recent and extensive hierarchical classification has been created by Dahlstedt and Persson [44]. They base their classification on a first level of *structural*, *constrain*, and *cost/value* interdependency types. According to their classification, *structural* types, such as REFINED- TO or SIMILAR- TO denote structural cross-references. *Constrain* types are characterized by an inherent semantics (such as REQUIRES or CONFLICTS- WITH). Finally, *cost/value* interdependency types can be used to describe process-related influence on other artifacts. Dahlstedt and Persson also state that their classification can be used as a basis for further refinement tailored to an actual domain or project.

One such refinement is, for example, the one by Aizenbud-Reshef et al. [2] who detail the *dependency* relation and extend their description to very fine-grained link types, such as CLASS- IMPORTS- CLASS or METHODINVOCATION-CALLS- METHODDEFINITION. One of the most extensive surveys of possible refined link types in RE is contained in the *high-level traceability model* by Ramesh and Jarke [156] which includes about 50 different types of traceability links. However, these links are again not formally defined. This can lead to ambiguity and misunderstanding when the meaning of a link has to be interpreted. On the other hand, the high-level traceability model is also not universal. Such a universal, semantically unambiguous traceability model as part of a universal traceability scheme has yet to be found [109,156,193].

Paige et al. [141] describe an incremental process which can be used to build hierarchical and fine-grained classifications of links. In this process, they use model management operations as the core concept on which the classification is built. Consequently, this raises the level of abstraction from regarding a traceability link as just an indication that a model transformation happened. Instead, an operation, a sequence of operations, or the results thereof are represented in a more logical way. Hence, a sequence of model transformations which (in this context) would derive a set of new model elements could result in DERIVED- FROM traceability links, which also would be compatible with, for example, user-created DERIVED- FROM links. In summary, this approach seems to be a promising step towards the creation of sensible, project-specific traceability classifications which include both the requirements from the MDD and from the requirements traceability community.

Additionally, there have been proposals of operational semantics [2] and of formal definitions of traceability links [72]. While these are a good start towards a more thorough definition and understanding of link types, they mainly regard semantics as a way to assure constraints imposed by the traceability scheme and to react to artifact changes. Particularly with informal information, such as textual requirements, it is still hard to decide, what kinds of types, should be used to characterize traceability links.

Finally, the structure of traceability links also leaves some open questions. Most traceability link models use binary links to represent traces. In contrast to this, several people have defined traceability links as potentially n-ary [87,123,179]. Furthermore, Dick [51] expresses the need for more advanced concepts such as alternatives or conjunctions between different traceability links. These could also be realized as either n-ary traceability links or as a set of interrelated binary traceability links, which would again require an enhanced notation to express such interrelations.

In summary, there are many open issues related to traceability schemes, and the structure and semantics of traceability links, as well as advanced challenges, such as tailoring traceability schemes to make them usable, economically beneficial, and suitable for a particular project. It can be observed that in both communities there have been a lot of discussions about these topics. Yet, it seems that these discussions have been carried out mostly independently although the motivation and goals are quite common. If a general and overall consensus on these topics shall be reached, both communities

will have to join forces regarding this topic. This section only gave a brief overview, as a more in-depth analysis would go beyond the scope of this article. However, as a starting point, a detailed survey and comparison of existing proposals of metamodels and classification schemes should be created.

3 Working with traces

Traceability (even in MDD) does not come for free. In order to enable traceability in a development project different steps have to be taken. These steps are detailed in the following section. Also, even if a project's artifacts are traceable, what are the benefits? This question will be answered in the second part of this section as we discuss which particular goals can be achieved with traces.

3.1 Activities supporting traceability

Basically, there are four activities when working with traces: *planning* for traceability, *recording* traces, *using* them, and *maintaining* them. This section outlines these activities as Pinheiro [145] describes them and translates them to the perspective of MDD.

Planning and preparing for traceability goes hand in hand with the planning phase of a software development project. During this activity, traceability methods and supporting tools are selected. Additionally, the artifacts which will be created during the project and the detail of the traces which will be recorded for them are identified. The outcome of this activity is a traceability scheme as described in the previous section, as well as the supporting tools which have been initialized and configured accordingly. Walderhaug et al. [195] describe this step as defining the trace metamodel (which is an instance of the traceability metamodel specified by the tool vendor). Additionally, in MDD this activity can be seen as part of a general setup of all required tools and artifacts in order to perform automated transformation. This leads to tools providing an infrastructure as introduced by Bézivin et al. [20], Blanc et al. [21], or Vanhooff et al. [189]. These tools use models (sometimes called mega-models) to define the relationships between models, e.g., the metamodel of a model or its source and target models. Using a mega-model, transformation chains can be set up, including traceability models. These tools can be roughly compared to makefiles, in which the whole build process is described.

Recording is an activity usually performed as part of every regular software development activity which leaves traces on the artifacts. These traces are made persistent by populating the tools and data structures prepared earlier. Regarding requirements traceability, this can be done either *on-line*, in which case traces are stored automatically by a tool as a by-product of the development activity. Or it can be done *off-line*, which means that traces are recorded automatically or manually after the actual development activity has been finished. In this case, the recording activity should be performed as soon as possible. Traces that are recorded later are usually more imprecise than those recorded immediately [36]. Additionally, only those stakeholders directly involved in the creation or modification process of the artifact can successfully record the traces [16].

In MDD, traces can be produced *on-line* as a by-product of transformation activities. This has even been specified in the OMG MOF Query/View/Transformation (QVT) standard for models [133] and demonstrated for the Atlas Transformation Language (ATL) by Jouault [98], for Kermeta by Falleri et al. [64], for FUJABA by van Gorp and Janssens [183] and in a more general notation by Vanhooff et al. [190]. The same applies to model-to-text and model-to-code transformation, as described by Oldevik and Neple [137] and Olsen and Oldevik [138]. Other methods of automatically recording traces are the approach of Mäder et al. [114] based on subsequent model editing activities and the approach of Wenzel et al. [197] based on model differences.

Yet, as discussed in Sect. 2.2.3, there are traces which are not merely direct by-products of these kinds of model changes and which have still to be recorded explicitly.

Using the traces can be part of several software development activities. Depending on the usage scenario, the data describing the traces is accessed in order to produce reports, to find relevant information, etc. Usage scenarios for traces both in requirements traceability and in MDD will be discussed in detail in Sect. 3.2.

Maintaining finally, is an activity resulting from either a structural change of the development process, or from an error or omission in the trace data which has been detected, for example, while searching for a trace. It differs from a usual *recording* activity in that the supporting traceability tools and metamodels have to be reviewed and adjusted, resulting in a new iteration of *planning and preparing*, or in that missing traces have to be identified and errors have to be corrected following the *recording* activity. The latter can also be seen as a separate *enhancement* activity [62].

These activities are also summarized in the workflow depicted in Fig. 5. Note that these are supportive activities performed as part of all steps in the software development process [142] and there is no simple relation between activities of a software development process and those four

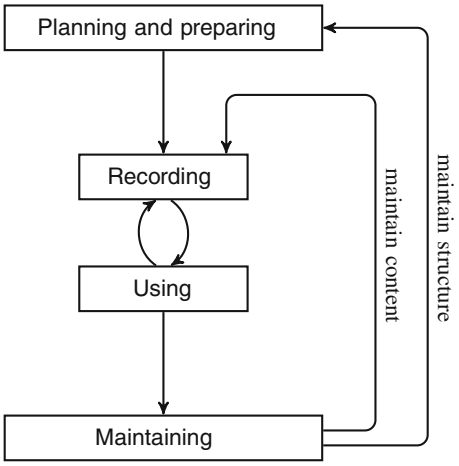

Fig. 5 The workflow of activities supporting traceability

activities. So, while traceability planning can and should be performed as part of the software development process planning phase, recording, using, and maintaining traces are activities performed whenever they are needed during the software development process.

Further on, traceability activities are not dependent on any particular software process model. Initially, traceability has been mandated by military regulations and thus, it has been part of a waterfall development process. Today, however, traceability has gained in significance particularly in iterative processes, where changes of artifacts have to be anticipated and managed during multiple iterations. Finally, in the past years, tools such as ECHO [107] and semantic wikis [47] have emerged which support a lightweight traceability approach usable in agile processes.

All in all, traceability and its supporting activities currently are not standardized. The entanglement between the traceability and software development activities heavily depends on the approaches used to record and use trace data. And so far, there is only little to no guidance for practitioners for selecting and combining these approaches into standard procedures.

3.2 Making use of traces

Recording and storing traces only makes sense if the traces can be used later. In fact, the project's traceability goals should generally drive the activity of recording traces [2]. The next two sections give an overview of those goals, or usage scenarios, as they are reported in the literature of requirements traceability and traceability in MDD, respectively.

3.2.1 Trace usage in requirements traceability

For requirements traceability, several contributions and empirical studies about how traces can be used have been published by Gotel and Finkelstein [74], Ramesh and Edwards [155], von Knethen and Paech [193], Watkins and Neal [196], and Wieringa [200]. We have already aggregated this information and complemented it with some reports from more recent contributions and with the roles of the different users of traces [201]. For reference and comparison to the usage of traces in MDD, we summarize our findings in the following list of uses for traces.

Prioritizing requirements If requirements can be traced back to originating goals or mission statements, they can be rated in terms of risk and priority [155]. Based on this rating, releases, iterations and development cycles can be planned.

Estimating change impact If the interdependencies between all artifacts are documented and maintained, proposed changes of requirements can be traced to necessary changes in downstream artifacts, supporting implementation time and cost estimations [74,155]. Additionally, impact on the quality, such as stability or security, can be predicted [16].

Proving system adequateness Traces can prove to the customer that every requirement is realized by the implementation with reasonable quality and that no extra features, which were not required, are included in the implementation [16,74,155,196].

Validating artifacts Traces can also be used to validate an artifact in several ways. In audits or reviews, traceability links can be used to ensure that a downstream artifact satisfies the upstream specification. In addition, inconsistency, incompleteness, and other defects can be detected early [155,196].

Testing the system On several levels of software testing, traces can be used to consult the right parts of the specification in order to generate or reference appropriate test data and test specifications, and to guarantee that all requirements have been covered by the tests [16,74,196].

Supporting special audits If a system is examined for special characteristics, like in a security audit, traces can help to identify the critical elements of the system [74].

Improving changeability The ability to determine change impact and to follow potential changes of the requirements down to dependent requirements and artifacts improves changeability and maintainability of the artifacts, and of the software [16,74,155,196].

Extracting metrics Detected inconsistencies, incompletenesses, defect rate, change rate, and several other metrics can be extracted and evaluated by analyzing captured traces [74].

Monitoring progress From a management perspective, traces can be used to monitor the status of the requirements during development as they progress from planned to implemented and tested [16,74,105,155].

Assessing the development process Since traces contain a log of the events occurred during software development, the development process itself can be followed, evaluated, and revised using metrics and other advanced information [74].

Understanding the system Traces can help to understand systems from different points of view [163], to pull together fragmented information [74], to identify crosscutting concerns [181], and to quickly look up the information just needed [16].

Tracking rationale For every artifact produced in the software development process, it can be determined, how and why it was created or changed. This can help to recall or reconstruct design and implementation decisions [16, 155].

Establishing accountability If a particular artifact or decision of the system cannot be understood or reconstructed, it is necessary to determine who created or modified the artifact and who made the decision. Traces can help finding answers for these questions [155].

Documenting reengineering Traceability links can also exist between systems and be followed back to other systems, such as legacy systems. Thus, relations between legacy functionality and features of the new software can be documented [58, 199, p. 358].

Finding reusable elements If a new system is developed and some requirements already have been implemented in an existing system, these requirements and the dependent design and implementation elements can be identified and reused [152, 199, p. 358]. This is particularly sensible for creating a new variant of an existing system in a product line [16]. In addition, requirements with a high cohesion could be identified together with their downstream artifacts and extracted as features or services [106].

Extracting best practices Finally, the trace database can be analyzed and audits can be performed [34] in order to identify best practices and to make them repeatable [96].

All of these activities would be possible, assuming that the traces needed for the scenario are complete, correct and up to date. The problem is that for most scenarios it is currently unclear which traces are actually needed for the single scenarios. Additionally, despite the many different usage scenarios for traces reported in the literature, most descriptions of traceability methods, techniques, or experiments only concentrate on a very small subset of those scenarios. This contradicts the goal of traceability schemes and models which aim to be universal, as discussed earlier.

Consequently, it is an open issue to match trace usage and traceability schemes, and to provide guidance to limit and fit traceability schemes in a way that they match a project's required usage scenarios for traces. One of the most urgent questions is, which requirements a single scenario imposes on the other activities (in particular planning and recording) in the traceability process.

3.2.2 Trace usage in MDD

As the history record of MDD is shorter than that of requirements engineering, an extensive empirical overview on how to use traceability links is still missing. Yet, by analyzing single contributions in that area, both usage scenarios similar to the ones above, and scenarios specific to MDD can be identified:

Supporting design decisions By consulting traceability links, different design alternatives can be evaluated and thus, the best alternative can be identified easier [25].

Proving adequateness/Validation By analyzing traceability links transitively through the different models and thereby establishing end-to-end traceability, it can be analyzed if a requirement is fulfilled and if an artifact is up to date [141]. Also, coverage and orphan analysis can help to identify problems with models or the MDD toolchain itself [195]. Furthermore, in post-model-traceability, different checks (e.g., to detect orphans) can be performed [138].

Understanding and managing artifacts Traceability helps to understand the many dependencies that exist between MDE artifacts and help control the wealth of different artifacts in the development process altogether [1, 30, 141, 195].

Understanding and debugging transformations As traceability links can be produced as a by-product of model transformations, they also provide valuable information which can be used to understand and debug transformations [8]. Additionally, it can be checked that all relevant parts of a source model are considered by a transformation [138].

Deriving usable visualizations In model transformation, the source model's diagram is usually laid out in a way the user knows. The target model, however, is not annotated with layout information, so automatic layout algorithms which perform bad in terms of usability are applied. In this case, traces can help to map the source model's layout information to the target model [186] and, thus, to produce more usable diagrams.

Change impact analysis As described in the previous section, traces can be used to simulate changes to an artifact and to explore their effects on the other artifacts [195]. In addition, however, in the area of formally defined models, changes can be propagated automatically using traceability links [26, 192].

Synchronizing models Similar to propagating changes, traces can be used to keep models for different purposes synchronized and consistent. Fritzsche et al. [66], e.g.,

describes an environment in which an application model and a performance model are synchronized in order to provide developers with a tool to try out different designs and analyze the impact on performance. Additionally, traces can be followed to upstream artifacts in order to achieve reverse engineering [195].

Driving product-line development finally, has received much attention recently [10,169] as an application where traceability can help to manage artifacts, variation points, etc.

While the why and how of some usage scenarios like transformation debugging are described quite extensive, others, such as understanding and managing artifacts need further elaboration in order to be understood and implemented in usable tools. As with requirements traceability, most publications describe their notion of traceability and their custom trace metamodel only in the context of a single scenario. Unifying those metamodels and integrating the different scenarios in usable workbenches and toolsets is work to be done.

3.3 Representation and visualization of traces

Generally, in order to be able to work with traces and to use them, user interfaces, representations, and visualizations are needed. There are several representations, some or all of which are implemented in current requirements traceability tools. Essentially, according to Wieringa [200], they can be categorized in matrices, cross-references, and graph-based representations. As in MDD models are the central element, graph-based representations are the norm. Figure 6 illustrates the three different representations by example.

3.3.1 Traceability matrices

A traceability matrix is a two-dimensional grid which represents traceability links that exist between two sets of artifacts, such as requirements, design elements, etc. The rows and columns of the grid are associated with the artifacts, and marks at the intersections represent the existence of a link. In Fig. 6a, for instance, a black box means, that there is a directed link from the left artifact to the artifact at the top. This form of visualization is the most traditional one in the field of requirements traceability.

While early forms of traceability matrices only provided support for a single type of mark representing the existence or non-existence of a link between two artifacts, traceability matrices today can be enhanced to include additional information about artifacts and links [145]. For example, an artifact in the matrix is usually referenced using an identifier but modern user interfaces can provide popup windows directly showing an artifact's meta-information or content, if needed. Furthermore, link types or other information could be encoded using different colors or symbols [55].

Even for non-technical users, traceability matrices are easy to understand and use in simple scenarios, such as recording or checking the existence of single links between artifacts [200]. However, for real-world projects, traceability matrices can become very large and unreadable [31]. The content of the artifacts is usually not displayed in the matrix, but referenced using unique identifiers. This separation and the two-dimensional nature of the grid, makes it difficult to follow links across several artifacts recursively. Finally, it is almost impossible to represent n-ary links in a two-dimensional traceability matrix in an understandable way.

3.3.2 Cross-references

Traceability links can also be expressed as cross-references. A cross-reference in its most trivial form can be embedded and represented in natural language (as, for example, in "*see the SRS revision* 1.2*, section* 3.14") or annotated using cross-referencing features of the word processor used. Ideally, such a cross-reference is presented to the user as a hyperlink, which he can click to navigate along the link.

An alternative view is illustrated in Fig. 6b. Here, the references are not an explicit part of the artifact, but stored as part of the artifact's meta-data. This enables displaying incoming links (from other artifacts referencing the artifact considered) as well.

Cross-references can be intuitively understood by any reader. However, the view on traceability links is very narrow in this representation: the user only sees outgoing—and

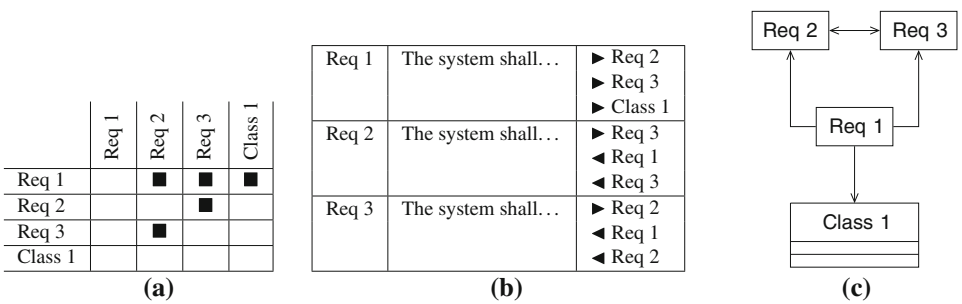

Fig. 6 Illustration of the most common traceability link visualizations. **a** Traceability matrix; **b** cross-references; **c** graph-based visualization

possibly also incoming—links of just one artifact. In contrast to traceability matrices, however, the links, the content of the one artifact, and other information associated with it, is usually displayed at the same time. So, compared to traceability matrices, the user is shown more local information at the cost of being shown fewer (global) links. Also, in the case of many references from and to one single artifact, the user could have problems with the decision, which one is relevant for his particular goal. Furthermore, visualizing n-ary links in a sensible way is almost impossible.

3.3.3 Graph-based visualization

In MDD, traceability links are often expressed as part of a model, and even in the requirements domain, traceability schemes are usually described as metamodels, as discussed before. If the artifacts are interpreted as nodes and traceability links as edges, the models can be visualized just like models from other domains in a graph-based notation—in fact, a traceability matrix can be interpreted as the adjacency matrix of such a graph.

As with matrices, general artifacts can be referenced using unique identifiers, as in Fig. 6c where the content is separated from the traces. This kind of diagram saves space to present more global information. However, a graph-based representation could also include part of, or all of the content of the artifacts displayed as. For example, a textual requirement could be contained in the diagram element representing the requirement. In this variant, the user can be shown several artifacts including their content and traceability links between them. Also, different link types, and even n-ary links can be visualized.

In the domain of MDD, most artifacts are model elements themselves which can be represented in a very concise way, as diagrams are the natural medium of communication for these artifacts. Thus, existing diagrams and representations can be reused and complemented with traceability links to visualize them. As a consequence, graph-based visualization is by far the most common form of representing traceability links in this area.

In requirements traceability, the flexible possibilities of graph-based visualizations is used by different tools: The PRO-ART environment by Pohl [148], for instance, contains the *Star View Dependency Browser*, a graphical front-end which shows an artifact surrounded by its directly dependent artifacts. The Star browser also allows to navigate to these adjacent artifacts, in which case the artifacts with dependencies on the newly selected artifact are added to the view. This is a quite sensible way to browse a set of traceability links. Regarding the notation, there is, however, no common agreement or standard, mostly because the variety and informality of different artifacts is not suitable for a simple, yet precise notation. The tool TOOR [146], for example, simply uses identifier labels interconnected by arrows. Likewise, Sinha et al. [167] use simple box and line models and Mohan and Ramesh [120] use icons together with labeled arrows. All in all, requirements traceability graphs usually are just plain box-and-line diagrams.

Also, graphs are not limited to visualize the dependencies between artifacts only: Kwan et al. [104] and Trainer et al. [180], visualize different aspects of social structures, project contribution, and formal or informal communication between the stakeholders using graphs, which can provide valuable information, in particular regarding pre-RS-traceability.

Despite all of these advantages, models and diagrams are often not as intuitive to ordinary users as matrices or cross-references, as they can often become huge and thus, hard to understand [86]. In addition, just adding traceability links to existing model visualizations also adds complexity and makes the diagrams hard to read and the models difficult to understand. Consequently, diagrams have to be prepared carefully in order to be understandable and useful, and elements and links have to be filtered accordingly.

3.3.4 Miscellaneous visualization approaches

Beyond these three traditional types of visualization, there are only very few prototypes which try to visualize traces in a different way:

Marcus et al. [118] propose a map consisting of colored and labeled boxes to show links from and to a single artifact which is similar to the hyperlink representation. Antoniol et al. [14] illustrate feature evolution and overlap using a 3D skyline-like view. Their visualization can be categorized as a chart highlighting feature-related aspects. Finally, Cleland-Huang et al. [38] propose some visualization prototypes addressing the aspects of dynamic trace retrieval tasks.

Regarding models, von Pilgrim et al. [187] propose a 3D enhancement to trace visualizations for model transformation chains: They use layered planes to visualize traceability links between different levels of abstraction. They show that the third dimension helps to present more content at once and to group related information together.

3.4 Usability of trace representations

As we have stated earlier [201], an essential part of usability (as defined in ISO9241) is the suitability for the user and task. These two aspects have been widely ignored in trace representations and visualizations up to now.

No matter in which domain, the different usage scenarios for traces demand for different user interfaces and searching facilities: A project leader who monitors the progress of the software development process only needs an up to date, rather static report, while a developer who wants to

understand the system needs very flexible and ergonomic search and navigation features across all media.

The area of searching and browsing traces in an efficient, user-friendly, and adequate way with respect to the roles and goals of the stakeholders has almost not been investigated up to now. Compared to the number of approaches addressing trace recording, there are only few reports on how to search or navigate traces. One reason for this is that most researchers often consider using traces as trivial, which it is not [118]. So, the challenges for future research are to provide an easy, appropriate, and usable access to traces in order to efficiently make use of them.

A first step into this directions would be the identification and aggregation of the requirements which the different usage scenarios impose on trace searching, browsing, and presenting facilities. These findings must then be implemented in future tools in order to make traces more usable.

4 Traceability in practice

So far, this article has described a research perspective on traceability in theory: starting with the term definitions, we have then described different schemes and metamodels, followed by a description of the different activities relating to traceability. The practice of traceability, naturally, is a different topic. In order to investigate this further, we start with a description of what motivates traceability in industrial settings and what is the state of the practice regarding traceability. Then, we identify and classify several limiting factors preventing the wide adoption of traceability practices.

4.1 The state of the practice

In its beginnings, requirements traceability was driven mainly by obligatory regulations such as *DoD 2167a/MIL-STD-498* for US military systems [95], *IEC 880* in nuclear system, *CENELEC EN 50126*, *EN 50128* and *EN 50129* in railway system in Europe, and *DO-178* and *DO-254* in aeronautic systems [3]. Later, quality standards which recommended traceability such as *IEEE Std. 1219*, *ISO 9000ff*, *ISO 15504* (*SPICE*), and *SEI CMM/CMMI* have gained more and more attention.

Today, traceability has become a recognized attribute of software quality: In a recent report, Rummler et al. [162] describe that missing traceability has been explicitly criticized in a quality certification audit of a large software development company. They also report that a large fraction of customers of the same company have complained about missing traceability—an observation also made in another environment by Hofmann and Lehner [88].

Consequently, there are two main motivations for software development companies to implement traceability practices: Either they are forced to do so, because they operate in one of the domains where regulations demand for traceability, or they are implementing traceability practices because they want to pass a quality certification or because their customers ask for it. A third, very rare, motivation for a company is that they see real benefit in traceability for their own work—for example, because traceability makes them produce higher quality software more efficiently.

Similarly, Ramesh [153] and Ramesh and Jarke [156] have identified two different groups of people concerned with requirements traceability in their empirical investigations: *low-end* and *high-end* users of traceability. According to their findings, low-end users tend to perceive traceability as a necessity imposed by regulations, sponsors, or customers. They see the task of recording traces as cumbersome and mostly useless and thus, they do not invest much effort in setting up or following policies, acquiring and customizing tools, or building environments in which traces can be used efficiently. Consequently, they do not make use of traces which in turn strengthens their notion of traceability being useless.

In contrast to this, high-end users are convinced that traceability is beneficial and set up tools, technologies, strategies and policies in order to record and use traces. Also, they do not apply traceability techniques blindly, but evaluate alternatives, develop usage scenarios, optimize benefit, and tailor traceability tasks to their needs. Over time, they build up a set of success stories, best practices, and general experience which they use to further optimize their traceability process.

Requirements traceability in practice is usually managed in requirements management tools. A detailed table describing the features of most of the available tools is maintained by the INCONSE.[4] The tool most commonly used to manage requirements and to record traces in the industry is DOORS [7]. Most other commercial tools in that area work more or less in the same way: Basically, these tools organize requirements and other fine-grained artifacts as attributed tree-lists which they present to the users using a spreadsheet metaphor. Every artifact is assigned a unique identifier and can be annotated using a set of attributes which can store metadata, such as priority, status, or risk. Additionally, artifacts can be linked to each other and to external files and thereby make them traceable. Traceability links can be visualized in a traceability matrix, as cross-references in the table-view or in a model- or graph-like diagram. Most tools provide the possibility to sort and filter traceability links and to generate reports. However, most tools lack a customizable set of link types and support for the different usage scenarios of traceability.

[4] http://www.paper-review.com/tools/rms/.

In high-end environments, these tools have to be—and usually are—customized in order to fit into the development process and culture, and generally, most traceability success stories are related to a lot of customization and development effort. If some time and effort is invested, requirements management tools can be integrated into the environment as reported, for example, by Alexander [4,5], Arkley and Riddle [16], and Dick [51]. Alexander reports that an enhanced and customized version of DOORS greatly improved the acceptance for traceability in a railway control system project. At the same time, Neumuller and Grunbacher [129] observe that traceability practices are mainly used in large companies and show that also very small companies can apply traceability beneficially. To achieve this, they have developed a specialized tool concentrating on the particular problems of that company.

Another case of successful introduction of traceability practices is reported by Asuncion et al. [17]. They describe a custom web-based end-to-end traceability tool which is used to store and manage traces for all projects in their company. Additionally, they state that technical, economical, and social aspects of traceability have to be addressed together in order to introduce traceability practices successfully, and they list a number of concrete guidelines on how to reach success in that area.

Similarly, Kirova et al. [100] describe their experience of evaluating traceability methodologies, developing a custom tool which integrates with other tools in their organization, and using this tool to support their processes. One of the successes they report is that the creation of several reports and reviews, which took up to two days without the use of traces, only took seconds when traces were used.

Regarding MDD, traceability in practice is not very widespread. Although there are a lot of research prototypes (e.g., [8,98]) and case-studies (e.g., [162]), there's no tool for MDD, which is both usable in industrial environments and able to generate traceability links. The only notable industrial application of traceability is the bi-directional synchronization of models and code in roundtrip-engineering UML tools (such as IBM Rational Software Modeler). Code produced with this kind of tool usually is annotated with tokens (which can be interpreted as traces) enabling the tool to track changes in the code.

Galvão and Göknil [67] present a survey of MDD traceability approaches and compare them using different criteria including tool support. A couple of these approaches provide native tool support, but in some cases (e.g., in the transformation languages ATL [98] or Kermeta [64]) traceability is not supported natively and is only achievable by extending the original tools.

Another common problem is that most stakeholders are not familiar with tools to record, manage, and use traces [4]. Also, a study by Gills [69] shows that each company or even each project creates its own trace metamodel almost from scratch. These factors also lead to higher effort and frustration. This affects low-end users in particular, making them feel vindicated even more in their view of traceability as a useless encumbrance. So despite of some punctual success stories where traceability is used and regarded as beneficial for the project, traces are usually neither maintained nor managed in a controlled and disciplined way. Traces, which have to be identified mostly manually, are recorded half-heartedly if at all, and almost never consulted or used as a support for software development activities.

The next section analyzes the situation in more detail and identifies the factors constraining the wide acceptance and application of traceability practices.

4.2 Limiting factors

Both in MDE and in requirements engineering, the commitment to traceability in practice is quite low. But what hinders the application of traceability practices in the industry? Trying to find an answer to this question, one can identify several limiting factors affecting traceability. Those factors can be classified as

Natural As shown in Sect. 2.3, there is not yet a common understanding of a complete, ultimate and well-defined traceability scheme or metamodel, and it is possible that it might actually never be found. This imprecise and incomplete nature of traces leads to a natural limitation to traceability, as with a complete, well-defined, and formal theory of traceability, many practical problems of traceability could be addressed quite effectively. Brooks [28] describes natural (or inherent) limitations in the context of software development as essential difficulties, and argues that there is no "silver bullet" for such limitations. Projected onto traceability, this means that it is impossible to capture traces completely [176], because the methods of human work are imprecise themselves, and because they incorporate implicit, social, and tacit knowledge [174], which can neither be formalized completely, nor be recorded and managed completely. Even in the rather formal modeling domain, some aspects remain which cannot be captured in explicit traces.

Technical Even if the previous hypothesis is falsified and a precise, semantically well-defined and complete definition of a traceability metamodel can be found, there are technical limitations that will remain: Due to the commonly used informal notation of pre-RS artifacts, requirements, model element documentation, and descriptions of context, decisions, etc. [71], traces cannot be identified completely using automatic processing without human intervention. Even in the relatively formal context of model

driven software development, establishing and maintaining traceability links is still an issue [65,173]. Human interaction is always required in order to determine the semantics of information and to record meaningful traces properly. This task, however, is not yet supported well enough by tools. Furthermore, as software is usually developed using several independent tools, lack of integration is also a source of problems: Most tools do not integrate well with tools from different vendors or tools for different purposes. This is particularly true for tools not directly related to software development, such as communication tools which become more and more important, as software development becomes a more and more global process. Consequently, recording traces is cumbersome and erroneous work. Regarding MDD, where it is technically possible to automatically record traces as a by-product of the transformation process, there are still issues, as setting up tools and configuring transformations for traceability is not very well supported and is thus also difficult and error-prone. Additionally, after the traces have been recorded, they have to be kept up to date, as requirements and artifacts change. If too little effort is put into maintaining the traces, links will become outdated. Again, support for the integration of different tools that could help in maintaining traces is generally poor.

Economical It is not easy to measure the return on investment (ROI) of traceability [142]. Investing in traceability feels more like investing in an insurance that is not regarded as indispensable, which is why the software development industry currently does not invest sufficient resources in applying traceability practices. Some of the consequences are lower changeability and higher maintenance costs. Yet, project leaders have doubts in the benefit of traceability measures and tend to neglect traceability. This is even more the case, as software development projects are usually short of resources and time, and management usually is concerned most about short-term satisfaction of the plan. Recording too much traces bears the risk of documenting traces that are difficult to manage and that will possibly never be used. Additionally, evidence of traceability benefits is only anecdotal so far. There is no sound empirical proof in which cases traceability is beneficial for a project or a company.

Social Another issue arises, if recording traces requires human interaction which is mostly the case in requirements traceability: Even if project management is convinced of the potential of traceability and if measures, tailored to the needs of the project, are taken in order to ensure traceability, the proper recording of traces can be hindered by missing personal motivation of the project's staff. Arkley and Riddle [16] repeat an observation by Gotel and Finkelstein [74] that the group of recorders and the group of users of traces are usually distinct. This leads to a low motivation of the recorders and thus to a lower quality of traces. Additionally, an experiment by Hayes and Dekhtyar [81] shows that semi-automated processes or tools can be weakened by people who might worsen tool results instead of improving and complementing them. Among the reasons for this are distrust or false trust in the tools and the sheer amount of candidate results retrieved by the tools, as well as other factors such as missing motivation or the lack of understanding of traceability.

In order to improve the adoption of traceability practices in the industry, research has to concentrate on all of these limitations. The efforts made so far are described in the next section.

5 Recent research: overcoming the limitations

This section presents the state of the research trying to overcome the different limitations described in the previous section. First we will briefly discuss, how natural limitations are addressed. Then, we will describe research related to technical limitations, which can be further classified in integration issues, recording issues, and maintenance issues. A discussion of approaches to economical and social limitation concludes this section.

As stated in the introduction, it is difficult to clearly separate recent research publications into requirements traceability and model traceability. Therefore, we chose to use the classification given in Table 2 which is based on the artifacts between which traceability is established. In this context, *non-model*-artifacts denote, for instance, requirements, formal specifications, test cases, and code—although some of these artifacts also could be considered as special kinds of models. *Model*-artifacts include UML- and MOF-based models as well as other forms of models, such as concept models.

We chose this classification because it reflects the parallels and interdependencies of the RE and MDD domains. Firstly, the table shows that some approaches have been similarly applied in both fields while other solutions only exist in one of the fields. This can be a starting point to develop new ideas. Secondly, we can see that there is a large set of approaches trying to bridge the gap between both fields by interrelating non-model- and model-artifacts. However, most of these approaches are rooted in the RE field, and joining forces of both fields could further improve the current state. Finally, we can identify cross-cutting concerns such as economic aspects, which are subject to both fields and which is another opportunity to work together on new approaches.

The essential approaches listed in Table 2 are described in more detail in the following subsections.

Table 2 Classification of research

Section	Non-model ↔ non-model	Non-model ↔ model	Model ↔ model
5.2 Trace integration			
5.2.1 Tool integration	Integrate using common representation (Hypertext, Hypermedia, XML) [116,123,127,139,164,165]	Regard non-model-artifacts (e.g., requirements, code) as models [108,137] integrate using common API [77,168,195] integrate using common representation (Hypertext, Hypermedia, XML) [116,123,127,139,164,165]	Not as much of an issue in theory, as long as the same meta-metamodel is used. prevent technical coupling of traces and models through model weaving [19,102] integrate using common API [103,195] model infrastructures (e.g., AMMA [20], MODELPLEX)
5.2.2 Specialized tools	Trace capture and management tools: Process-centered (PRO-ART) [147] Object-oriented (TOOR) [144,146] Rationale-centered (REMAP) [119–121,154] Code-centered (SCE) [123,130]		No explicit traceability tools, only plugins to existing modeling tools
5.3 Trace recording			
5.3.1 Rule-based, structural	Based on program analysis [61] based on formal specification code generation [161]	None	Based on model-differences [197]
5.3.1 Rule-based, linguistic	Based on common keywords [99] based on grammatical properties [32] using machine learning [76]	Based on grammatical properties: requirements to UML [97,170,172] requirements to concept models [177,178]	None (not applicable as linguistic approaches are based on natural language)
5.3.2 Based on information-retrieval	Between documentation and code [13,117] between requirements and requirements [82,125,126] between requirements and testcases [46,112,113]	Between requirements and model elements [40,46,112,113]	None so far
5.3.3 Fully automated	Based on automatic transformation from natural language to formal logic [68] based on code synthesis [198]	Model generation from natural language similar to rule-based linguistic approaches (e.g., [91,92]—but do not explicitly mention traceability).	Based on automatic augmentation of model transformations with trace generation code [98]
5.4 Trace maintenance	Based on events/triggers [33,36,39]	Based on rules [124]	Based on rules [191,192] based on link semantics [2] based on events [43,114]
5.5 Economic aspects	Tailor traceability to the project's needs [59,62] value-based/cost-benefit aware traceability [84,85] provide immediate benefit to management [105] use different traceability methods for links of different importance [41] research towards easily tailoring traceability [97,115]		
5.6 Human aspects	Traceability needs to support the user and provide immediate benefit (e.g., [4,16,58,79,175])	Benefit exists (e.g., model-code-synchronization, roundtrip engineering, artifact consistency [128,160]), but applies only to structural models. Exception: synchronization of state models and code [49]	Recording traces is usually cheap and benefit exists (e.g., transformation debugging [8], model consistency [26,60,65,188])

5.1 Overcoming natural limitations

There are several approaches addressing the restrictions described in the previous section. Concerning the natural limitation of traceability, there is, as discussed before, no silver bullet. Some basic topics of traceability, however, are not thoroughly investigated and accepted yet. There is an ongoing discussion about traceability schemes and the nature of traceability links in the research community as described in Sect. 2.3. Even if this discussion will not produce any complete solution for the inherent problems of traceability, it will help to structure and understand problems with traceability better and to find a common view on it.

5.2 Overcoming integration issues

Integration issues can be addressed either by developing *integrative approaches* which adapt standard tools to a common basis, or by creating *specialized traceability tools* which provide user interfaces and methods explicitly tailored to traceability tasks.

5.2.1 Integrative approaches

Integrative approaches address the issue of medial heterogeneity: Traceability links between different products produced by different tools have to be stored and maintained in some way. Users that are already familiar with their development tools usually do not want to switch tools just because some other tool supports traceability. At the same time, almost every software development company uses a different combination of development tools to perform tasks such as requirements management, documentation, modeling, implementing, etc. Consequently, a non-intrusive, integrative way to manage, store and represent traces between existing tools has to be provided. As a side-effect of integration, some traces can be recorded automatically as a reaction to events or relations reported by the integrated tools.

In MDD integration regarding model-to-model traceability seems not to be an issue, because meta-metamodels serve as a common denominator for all kinds of models, and because transformation tools themselves have to provide an integrated view of the different models in order to work properly. An issue, however, is the technical coupling of different models (transformation model, trace model, source and target models, etc.). Traceability links between those models would technically mean to introduce dependencies which would lead to bloated models. To prevent this, Barbero et al. [19] and Kolovos et al. [102] propose the use of model on-demand merging or weaving of models and keep models decoupled otherwise. Moreover, considering pre-model- and post-model-traceability, there is the same variety of media and artifact types as in requirements traceability. Therefore, integration is also an issue for MDD. In this context [108] proposes a metamodel for traceability between textual requirements and models which regards single requirements as model elements and thus, making them traceable in the context of a model. Likewise, model-to-text traceability can be facilitated [137]. Bézivin et al. [20] use the term "MegaModel" to describe the modeling of an infrastructure (viz. a registry for models) and they present tools as part of the AMMA project to demonstrate how such an infrastructure could look like. Providing a modeling infrastructure is research topic of several projects, such as MODELPLEX.[5]

In a more abstract approach, the OPHELIA project [77, 168] defines a platform for distributed development tools interconnected by a set of middleware interfaces that enable tool integration. On top of this integration, OPHELIA defines a common service layer where crosscutting services such as metrics calculation, knowledge management, notification, and trace management are provided to the stakeholders.

In contrast to OPHELIA, where each tool still has full control over its artifacts and has to provide information about the artifacts by implementing a common interface, several approaches based on hypertext models such as HYDRA [149] try to integrate artifacts on the content level. It is even possible to integrate and cross-reference audiovisual media in hypermedia models [78]. Based on these models, the XML technology, which has gained in importance in recent years, also supports references and meta-data. This is utilized by several recent research projects to store traces in XML files. (e.g., [116,123,127]). XML is also used in the GENESIS subproject [22] of the OSCAR project, which uses a common repository to integrate artifacts: The GENESIS platform defines a distributed repository based on XML in which so-called "active" artifacts can be stored. An "active" artifact not only consists of its own content (like a piece of source code), but also carries information about its history and dependencies on other artifacts (such as requirements, test cases, or documentation). The goal is to provide an integrated view of dependencies not only within one project, but also of links to other projects of which components have been reused. In summary, while XML and hypertext models directly support representing and cross-referencing almost all types of data, most tools used to create artifacts only provide limited support for input and output of common or standardized representations. Therefore it is difficult to establish a working set of tools supporting both ordinary development tasks and trace management.

Olsson and Grundy [139] observe that different artifacts are not only created and managed in different tools, but also that information about the same part of a piece of software is present in several places, levels of abstraction, and in different representations. A key problem is to keep the

[5] http://www.modelplex.org.

information comprehensive and consistent. They present an approach which addresses these issues by providing views across parts of several artifacts and by showing change impact information to the user in order to manage possible inconsistencies introduced with changes. In order to integrate data from different sources, they use extractor modules which extract content from various documents and convert it into their own common data format.

Particularly in the development of embedded systems, several tools are usually combined in a tool-chain. Every tool is used for a particular development step and level of abstraction. Several automatic and semi-automatic transformations from one tool to another are usually carried out. However, these transformations usually do not produce or manage traceability links. Therefore, Königs and Schürr [103] have developed an approach to integrate these different tools by creating and implementing interfaces compliant to the Java Model Interface (JMI) standard. These interfaces are then used to reference artifacts as external model elements and to store and maintain references between these elements in a model repository. A similar approach has been described by Walderhaug et al. [195]: In their traceability solution they describe a central repository and an API which has to be implemented by different modeling tools such as model-to-model and model-to-text transformation engines in order to populate the trace model in the repository and thus, to share and integrate the traces. However, such implementations are often problematic, because they have to be based on mechanisms provided by tool vendors, which often do not comply with existing standards [182].

All of the approaches described above are only able to provide traceability links which are detached from the original tool in which the artifact has been created. Usually they use a custom user interface to show traceability links and related content of the original artifacts. The traceability links cannot be viewed and followed directly in the original environment: for example, it is not possible to follow a traceability link in a text file to a model, because the text editor usually is not able to render and handle links to models. This is why Sherba et al. [164,165] propose an approach called TraceM. This approach augments an existing hypermedia system [9] with explicit traceability features. Like in OPHELIA, the artifacts' content remains under control of the original tools, and only anchors and links are stored in an external repository provided by the hypermedia infrastructure. In addition, the original tools are enhanced using their own API so that they adapt to the hypermedia system and thus can provide a user interface for hypermedia links.

Each of the integrative approaches has been implemented as a prototype by the contributing authors. Yet, they are only poorly evaluated and thus, they are difficult to compare. In order to be really usable, traceability links have to be integrated directly into the user's workspace as, for example, proposed in the TraceM project, which requires standard interfaces in all tools involved. Standardization—although desirable—requires a lot of industrial commitment, which is not likely to be seen in the near future.

An alternative way of managing traceability links between different types of artifacts is the development of completely integrated tools, which are described in the next section.

5.2.2 Specialized traceability tools

A very extensive research prototype trying to overcome the limitations of commercial requirements engineering tools is PRO-ART, a tool proposed by Pohl [147]. PRO-ART is not designed exclusively to be a traceability or requirements management tool, but a tool in which the requirements engineering process can be defined and executed in a very fine-grained way. PRO-ART guides the users—software developers in different roles—through the requirements elicitation and specification process. As a by-product, it automatically captures traces in a schema based on the IRDS standard [11], which can be seen as an early manifestation of the current OMG Meta Object Facility (MOF) standard [134]. The tool provides automatic trace recording by logging all actions and process steps. The recorded traces can be queried and browsed later in different views.

The tool Traceability for Object-Oriented Requirements (TOOR) by Pinheiro et al. [144,146] aims to be an all-purpose traceability tool. As in Pohl's approach, the traceability metamodel in TOOR can be fully configured. Unlike PRO-ART, however, TOOR is not as process-oriented, nor is it restricted to requirements elicitation or editing of artifacts in the tool itself. Traceable objects and possible traces are defined using a declarative high-level object-oriented programming language called FOOPS. Specification and design elements can also be expressed directly in TOOR using FOOPS, which can be utilized to automatically capture traces from other artifacts to these elements. As a result of embedding and using a complete object-oriented language, TOOR provides a very flexible query mechanism.

In the REMAP project, which has focused mostly on recording design rationale, a model editor and a rule processor have been implemented by Ramesh and Dhar [154]. Design rationale can be modeled using an editor and a set of customizable rules can be applied to the model. This enables semi-automatic recording of rationale-related traces. The approach has been enhanced further to support product families [119] and Mohan et al. [120,121] also have proposed to use it for storing and managing distributed knowledge.

The Software Concordance (SC) Editor by Munson and Nguyen [123,130] directly addresses the transition from design to code. It provides an editor where code and arbitrary documentation based on the XML-format (e.g., requirements, architecture, or design descriptions) can be edited side

by side and elements of the XML tree and elements of the language abstract syntax tree can be interlinked. The approach is related to the literate programming approach described by Knuth [101]. In contrast to Knuth's approach, however, the code is not directly embedded into a linear text document, but a set of documents and the implementation can be referenced on a fine-grained level. The links are usually recorded manually by the user. However, the SC Editor also provides support for automatic handling of link evolution and implements a fine-grained version management of artifacts.

All of these tools provide their own editors, forms, and user interfaces. As a consequence, they are less intuitive to most users accustomed to standard development environments, but they provide a user interface adapted to support traceability activities very well. This does not only simplify manual recording of traces, but also enables automatic analysis of the artifacts for traces. All of the tools provide a more or less flexible way to record traceability links automatically using a set of rules. These rules are either hard-coded, like in the SC Editor, which incorporates special rules to handle artifact evolution, or they are freely configurable, as in REMAP and TOOR.

The next section discusses the topic of rule-based trace identification in more detail, together with several other approaches in this area which are not part of a full-fledged traceability tool.

5.3 Overcoming recording issues

Computers are generally not able to capture all traces automatically because of the informal nature of most of the underlying information. Nonetheless, several approaches have been created during the past 15 years which try to optimize automatic recording and identification of traces in different phases of software development. These can be classified in *rule-based approaches* which deduce traces by applying rules to artifacts and *approaches based on information retrieval algorithms* which can detect candidate traceability links between artifacts. In addition, there are some areas, where *fully automated traceability* is possible. Particularly in the domain of modeling several advances have been made. In order to compare how different trace recording approaches perform, several measures are used which are also described.

5.3.1 Rule-based approaches

Generally, the most important goal to improve requirements trace recording is to reduce costs by finding reliable and effective techniques capable of automatically discovering traces. One way to achieve this are rule-based approaches, which can be classified further in *structural* on the one hand, which are based on the structure and existing relations of artifacts and *linguistic* on the other hand, which analyze and process natural language texts and apply rules based on their syntax.

5.3.1.1 Structural Rule-Based Approaches Structural rule-based approaches apply rules to certain structural attributes of a trace model. This means that, for example, a type of traceability link can be defined as transitive: if a user uses this link type to record a link from an artifact A to an artifact B and another link from B to C, then the system is able to automatically derive and record a link from A to C because of the transitivity. Likewise, it is possible to derive attribute values of traces and to validate a trace model against a set of assertion rules. Most of the specialized tools described in the previous section include basic support for such rules. Some more advanced approaches are described here:

Egyed and Grünbacher [61] enhance the basis from which they derive links by following the approach of dynamic program analysis: They record program execution traces (these are—as described earlier—dynamic program behavior logs) from the execution of test-cases. They combine the results with a set of requirements-to-code traceability links, which have to be established manually beforehand and infer traceability links between requirements.

A tool developed by Richardson and Green [161] operates in the opposite direction: In a setting, where code is generated automatically from a formal specification, they observe and analyze the impact of small changes of the specification on the implementation. This impact can then be used to derive fine-grained forward traceability links from the specification to the implementation.

Finally, Wenzel et al. [197] describe a tool which is able to compare the version history of models, to derive traceability links, and consequently, to reconstruct the evolution steps of single model elements.

5.3.1.2 Linguistic Rule-Based Approaches The power of structural rules applied to requirements in natural language is rather limited, both because the user has to record most traces manually, and because the method can only be applied to very special scenarios involving formal structures. These restrictions can be eliminated, if the analysis of the structure is extended to the analysis of the language.

One of the most trivial linguistic rule-based approaches is to simply search texts for occurrences of common keywords and establish traceability links based on these keywords [99]. However, this simple approach can be complemented with natural language processing (NLP) techniques. In particular, part of speech taggers can be used to parse texts, to analyze the syntax, and to identify words together with their grammatical attributes.

The results of a lexical analysis can then be used to identify taxonomies of key terms in textual requirement documents. Cerbah and Euzenat [32] show that technical terms

in French usually follow a very regular local pattern based on which they can be identified and interrelated. This can be used to create traceability links between business object models, term dictionaries, and textual documents.

Spanoudakis et al. [172] use a part of speech tagger in combination with a rule-engine in order to derive traceability links. Unlike the relatively simple rules and links of Cerbah and Euzenat, however, they can derive different types of links between unstructured textual documents, use case specifications, and object models. The required rules can either be specified manually or inferred using machine learning [170]. The approach of machine learning has also been applied to the identification of traceability links between elements of a use case diagram and source code by Grechanik et al. [76].

Jirapanthong and Zisman [97] take the basic algorithm of Spanoudakis et al. [172] to another level: they first created a classification of descriptions and models used during the requirements specification phase of a product line system. Then they have derived a traceability metamodel and a rule-set which is able to derive traceability links between the different documents and models. They also show how traces can be used to support different tasks related to product-line development.

Strašunskas et al. [177,178] use an existing approach by Brasethvik and Gulla [23] to build concept models for information systems from natural language documents using part of speech tagging and syntax analysis. They show that these concept models can be used to derive traceability links which are useful for the calculation of change impact.

Most rule-based approaches require some form of human interaction in order to set up a basis for traceability. This may be either a set of traces recorded manually, or a basis of customized rules. Compared to recording traces completely manually, the costs are lower. However, flexibility is also generally decreased, as most approaches rely on aspects such as particularly structured artifacts or consistent use of terminology. From a modeling perspective, several of the rule-based approaches aim at establishing pre-model-traceability links.

5.3.2 Approaches based on information retrieval

A way to further reduce costs of manual interference concentrates on after-the-fact tracing by extracting requirements traceability links from textual documents using information retrieval (IR) methods [18]. These methods have been explored in the domain of internet search engines for several decades and have reached some maturity. The IR approaches are generally based on chunks of natural language text and usually do not rely on any structural properties of the input. In a nutshell, information retrieval returns a set of matching documents from a set of documents (called a *corpus*) for a given query. This is done by first extracting key terms from each document in the corpus and then computing the similarity between the terms of the query and the key terms of each document. A list of candidates is then presented to the user who has to decide which of the candidates are relevant to his query.

When IR principles are applied to trace retrieval, the corpus is built from the traceable artifacts. How the key terms are extracted from an artifact depends on the type and format of the artifact. If the artifact is a text in natural language, keywords can be extracted just like in ordinary information retrieval: by first eliminating all stop words (which consist of common words such as pronouns or determiners) and then stemming the remaining words (to avoid differences caused by different affixes to the same word). If the artifacts are models or code, identifiers, names, and other attributes can be used as keywords.

To do this, different mathematical models can be used. The most common models applied to traceability are Probabilistic Information Retrieval (PIR), Vector Space Information Retrieval (VSIR), and Latent Semantic Indexing (LSI). Different research groups have tried to compare results achievable with the different models: [13,82,112,117], to enhance them using different sources of additional information [40, 202] or to combine the different approaches [48].

IR-based trace retrieval has been proposed as an after-the-fact method for traceability link detection. The idea behind this strategy is to postpone the manual efforts invested in requirements traceability to as late as possible and thus, to reduce the risk of investing in activities which possibly turn out to be useless later. Thus, instead of recording traces, the list of candidate links returned by the trace retrieval algorithm has to be reviewed and each link has to be accepted or rejected by the user. This activity remains a manual task, which cannot be automated [80]. Yet, de Lucia et al. [46] and Lormans and van Deursen [113] have analyzed the impact of different strategies to post-process the list of candidate links in order to further reduce the amount of manual work.

The different approaches have also been implemented in the tools RETRO [83], ADAMS [45], and Poirot [110]. The latter has also been applied successfully to industrial settings [35].

Overall, trace retrieval does an acceptable job in the appropriate situation. The cost of recording traces is minimized to nearly zero, because manual intervention is only necessary when traces are queried and used. Naturally, the initial quality of the candidate list prior to the manual review is often poor. Consequently, the stakeholder who uses trace retrieval has to have some knowledge about the project, or at least the domain, as he has to decide if a candidate is truly a link. Furthermore, he cannot expect the candidate list to contain every single link, because some links could exist although the artifacts are not similar from the perspective of the retrieval algorithm. But all this does not seem to impact practical scenarios, as reported by Hayes et al. [82].

In an early approach to link identification using similarity measures, Natt och Dag et al. [126] have investigated the automatic detection of requirements relations and similarity by directly applying similarity measures to sentences based on common words. They conclude that their techniques, and fully automated processing of natural language in general, cannot replace human judgment. However, in their later work [125], they find that VSIR-based approaches can at least help in recovering pre-traceability links more efficiently: they argue that about 50% of the links can be identified using IR-based trace retrieval techniques in very little time. When compared to a manual keyword search for possible traceability links, trace retrieval yields a significant speedup.

Despite the relatively good ratio of cost and benefit, a common problem of most automatic approaches to traceability is semantics, both of the information analyzed and of the traces recorded. On the one hand, most approaches only work in settings, where all artifacts use the same terms consistently, or if there is a consistent thesaurus which is used. On the other hand, the semantics of traceability links itself is a problem. Most approaches are only able to retrieve pairs or sets of entities which are similar *in some way*, but the links' semantics cannot be derived in general. Therefore, alternative approaches discussed earlier still have to be used when semantically meaningful traceability links are needed.

5.3.3 Fully automated traceability

One area, in which fully automated traceability is already possible today—even including traceability link semantics to some extent—is the use of automatic generation, formal languages, models, and model transformation (cf. Sect. 3.1) [111]. Van Lamsweerde [184] reports several applications of formal specifications relevant to traceability: refinement of specifications, derivation of test cases, and extraction of specifications from source code are transforming activities which can produce traceability links as by-products. Of course, deriving a traceability link as part of a generative or transforming activity is rather intuitive: there is usually a link like DERIVED- FROM from the product to the source artifact.

Gervasi and Zowghi [68] have explored the automatic transformation from natural language requirements into formal logic. In their approach they analyze natural language requirements with a part of speech tagger, and then produce equivalent formulas in predicate logic. The source and target artifacts are stored in a database in order to make the transformation traceable.

A similar transformation from requirements to UML models is described by Ilieva and Ormandjieva [91,92]. While their model generation method could easily support traceability, they do not mention it explicitly.

There are, however, still open issues in this area. Most transformation tools are not able to output traces. Thus, new tools have to be developed which are able to record traces, and existing tools have to be enhanced, like the ExplainIt! enhancement to Amphion/NAV, a Fortran code generator [198]. Also, in the QVT domain, metamodels and transformation rules have to be explicitly enhanced for every project in order to make the transformations traceable. It is desirable that these enhancements do not affect the transformation specifications directly, because this would make them more difficult to read. Jouault [98] describes a way to automatically add traceability link creation logic to existing declarative model transformations in the ATL transformation language in order to decouple the transformation and trace generation aspects.

Yet, all of these aspects still need more and deeper research before reaching maturity. From a traceability perspective, it is a requirement for all future tools which can generate or transform artifacts related to software development to output fine-grained traces. If several of such tools are used together in a project, there has to be a way to integrate these traces. Additionally, although traceability links originating from these tools are helpful, they do not suffice, because they cannot capture informal dependencies between artifacts as, e.g., in the context of pre-model and post-model traceability. An overall standard format to represent, and work with, traces of different sources would be desirable.

5.4 Overcoming trace maintenance issues

While the recording of traces is an area of quite active research, particularly in requirements traceability, most approaches are not well suited to the evolution of the artifacts. In reality, iterative or even agile development processes, which anticipate change and evolution, are most common nowadays. To address this evolution and its consequences for traceability, only event-based approaches have been found as an appropriate means up to now.

Chen and Chou [33] have suggested to define trigger events and consistency checks for traceability links. They describe a framework in which decomposition of artifacts and traceability links between them are expressed in an object-oriented language. If an artifact is changed the change is propagated to all affected artifacts and the relations between them check the consistency of the artifact pairs, leading to notification events or exceptions in the case of a violation.

Cleland-Huang et al. [39] have proposed a publish-subscribe mechanism following the well-known *Observer* design pattern: an artifact depending on other artifacts registers this dependency at a central server. The evolution of artifacts can then be expressed as a series of change events. Stakeholders managing the artifacts of dependent artifacts can be notified of change events and instructed how to react. When an

artifact is changed, all *observers*—artifacts depending on the artifact—and their maintaining stakeholders, respectively, are notified and can check for potential changes of traceability links. At first, their idea has targeted performance impact analysis for requirement changes. Later on, however, Cleland-Huang et al. [36] have extended their approach to cover all artifacts.

Von Knethen [191] has proposed a set of models and a working environment [192] in the domain of embedded systems. The approach supports change impact analysis and semi-automatic propagation of changes through these models based on traceability links recorded previously. Similar proposals have also been made in the area of model-based development: Aizenbud-Reshef et al. [2] have defined operational semantics for traceability relations in UML models which are similar to the proposal of Chen and Chou [33], but also include reactive operations which can propagate changes automatically by modifying the impacted elements. Costa and da Silva [43] have presented a reactive traceability framework called RT-MDD, which is targeted at the domain of MDD and code generation. It uses current technologies from this domain, such as model queries and transformations, to update models and to keep them consistent with each other while change operations are performed.

Still, several of these approaches are targeted more at propagation of changes to artifacts and less at propagation of changes to traces and the maintenance of traces is often only a by-product in these cases. Apparently, all approaches addressing the maintenance of traceability links in the context of the evolution of artifacts use a variant of the same pattern: Existing traceability links are used to propagate changes from an artifact at one of the end points of the link to the artifact at the other end point(s). Thus, it is up to the stakeholder managing the affected artifact to maintain traceability links associated with it, as most approaches focus on updates on artifacts or refer to general changes.

Murta et al. [124] directly address the consistency and evolution of traceability links in the context of the independent evolution of an architecture description and its implementing source code. They present an open system which can be configured and enhanced with constraints guarding the consistency of traceability links and with so called policies—rules which are activated in order to propagate changes and to adapt the traceability model to the evolved artifacts.

Mäder et al. [114], finally, describe an extension to a modeling tool which records operations which the developer performs when he works with the model. If a chain of operations is recognized as a single operation at a higher level (e.g., the removal of an attribute and the creation of an attribute with the same name in another class could be recognized as a *move attribute* operation), the tool is able to maintain traceability links which are affected based on given rules.

A general problem regarding the maintenance of traces, however, is a scenario where, for instance, a link between two artifacts is created unknowingly: Take two existing requirements R_1 and R_2 which are not dependent on each other. If now R_1 is modified and a constraint is added which partially overlaps with R_2, a traceability link between the changed R_1' and R_2 should be recorded. This however is almost impossible, if the stakeholder who changes R_1 is not aware of this dependency. Ordinary traceability link recording is mostly concerned with recording causal dependencies, such as a requirement R_5 which is DERIVED FROM requirements R_1 and R_3. Trace maintenance is mostly concerned with keeping existing traces consistent. Hence, there is currently no approach other than tracing after the fact to address the problem of identifying unintended or unnoticed links.

5.5 Overcoming economical limitations

Economical limitations are mainly related to the ratios of cost, risk and benefit. While for requirements traceability, manual tasks, such as recording traces are the factor of highest cost, the most influencing part on economics in MDD is deployment and customization of the tools.

Regarding the former, recording the vast amount and detail of the traces described in some of the models discussed in Sect. 2.3 will never be achievable in practice. Therefore, it is essential to perform an analysis of those factors—cost, risk, and benefit—following the paradigm of value-based software-engineering. Egyed [59] names four aspects of traces that can be influenced in order to tailor traceability: precision, completeness, correctness, and timeliness. He also gives an analysis of their interdependencies and trade-offs. If the value-based paradigm is applied to traceability, cost, benefit, and risk will have to be determined separately for each trace according to if, when, and to what level of detail it will be needed later. This leads to more important artifacts having higher-quality traceability.

Heindl and Biffl [84] have investigated the effects of value-based traceability in practice and found that value-based approaches can reduce the effort for traceability by 65% in a first case study, compared to traditional approaches to traceability. Egyed et al. [62] continue exploring the effects of adjusting granularity using three case studies. They show that from an economical perspective, recording traceability links from requirements to classes is more optimal than to modules or to methods. They also found that high-value requirements often cover large portions of the code, so simply applying value-based practices and refining all traceability links from high-value requirements to code would not save

very much effort compared to generally recording all links on a finer level of granularity. They show, however, that a more adapted selection strategy focusing on classes shared between several requirements, cuts costs by 30–70% with no significant loss of traceability link quality. In their more recent research, Heindl and Biffl [85] have combined their findings with research results from other publications. As a result they have developed a model of traceability cost and benefit which they have shown in a case study to be useful as an instrument to estimate the return of investment of different tracing strategies.

Because economic aspects mainly influence management decisions, providing immediate benefit (see next section) to the management can also positively influence traceability practices. Lago et al. [105] reports that introducing traceability in a software development company has enabled a stand-in manager to create a project's progress report for the customer in under an hour.

One software tool addressing value-based methods in traceability has been proposed by Cleland-Huang et al. [41]. Their approach is based on the fact that every tracing solution has its advantages and disadvantages. In order to optimize the overall return of investment, they have come up with a system combining several traceability approaches. How traces are recorded is determined using risk analysis. When the traces are needed, the user queries a common front end which returns results aggregated from different data sources including manually recorded traces and traceability links retrieved dynamically using IR methods.

Regarding deployment and customization of tools, Dömges and Pohl [52], and Pinheiro [145] note that traceability methods and tools have to be easily customizable to the needs of the particular project which they will be applied to in order to lower traceability costs. More precisely, the goals of the application of traceability to the project have to be determined, and traceability schemes, methods, and tools have to be tailored in order to achieve these goals [193]. Likewise, guidelines and standard templates supporting the whole planning activity for traceability have to be developed. Two exemplary contributions in this directions are those by Philippow and Riebisch [115], who analyze the Unified Process for artifacts and traceability links, and by Jirapanthong and Zisman [97], who describe the planning steps for traceability in a product line development context.

In summary, during the past years, the cost of traceability has been reduced by applying technical solutions and by tailoring traceability processes. Simultaneously, the demand for traceability in practice has grown. This tendency will most likely continue in the future. This is why it is necessary to actively support this development by lowering costs, by emphasizing benefits further and by providing useful guidelines and templates.

5.6 Overcoming social limitations

Since trace recording of informal artifacts can never be automatized completely [59], the human factors affecting traceability persist. Hence, it is important to motivate users, to increase usability, and to convince users of the benefits that can be drawn from traceability. This can be done by developing approaches which provide users with immediate benefits from recording traces.

Arkley et al. [15] argue that in order to be accepted by the users, traceability activities must not be imposed on them, but be designed to support them in the way they work. In a later report, Arkley and Riddle [16] describe a success story, in which a project in the automotive industry has used an environment based on DOORS which has been customized in order to provide direct benefit to the staff. Even though trace recording required a significant amount of effort, the engineers were rather content with the solution.

Alexander [4] writes about his experiences customizing and using DOORS in an industrial railway system development project. He has made the experience that the use of plain paper, word processors, and email in industrial projects is not completely eliminated by providing an integrated requirements management and traceability environment. Nonetheless, the few simple and useful enhancements he had proposed, such as generation of browseable hypertexts, extraction of business terms, and a reuse mechanism for domain standards, have encouraged the use of requirements management and traceability tools.

Ebner and Kaindl [58] present a case study in which they have successfully applied traceability to a round-trip reengineering project. They describe that their solution has provided immediate short-term benefit which motivated the users.

In general, future commercial tools such as DOORS should regard the importance of traceability features and immediate benefit provided to the users. Tool providers should increase the efficiency and usability for these features. As Hoffmann et al. [87] argue, traceability acceptance will increase, if these tools are made more usable.

Some traceability research explicitly addresses tools and methods designed to provide benefit to the users in a specific usage scenario of traceability. Stone and Sawyer [175], for example, propose a tool based on LSI which is able to identify tacit knowledge in requirements. This helps analysts to pinpoint and document the implicit knowledge immediately, and thus, to improve the quality of the requirements. Haumer et al. [79] suggest to establish traceability links between real world scenarios documented using rich audiovisual media, and conceptual models of a system in order to facilitate better understanding and more effective reviews.

In the modeling domain, the human aspects of traceability are often not as much of an issue, because inter-model

traces can be recorded automatically. At the same time this is a domain where the immediate benefit of traces for the user is explicitly present: Most UML modeling tools support *roundtrip engineering* activities, consisting of code-generation from models and code-to-model reverse engineering components. Traceability links between both are kept as comments or annotations in the code. The environments automatically manage consistency and change propagation between models and code. Also, as described earlier, the use for traces as a means to perform transformation debugging and change propagation is quite obvious and graspable.

Yet, model-to-code traceability could still be improved, as consistency between models and code is usually limited to structure, leaving out behavioral aspects. Deng et al. [49], for example, describe a first step to maintain traceability links between UML statechart models and code implementing their logic.

Additionally, consistency between different models is also an area of current research in the modeling research community [26,60,188]. France and Rumpe [65] survey the advances and challenges in that area in more detail.

Following the relatively successful traceability approaches concentrating on consistency checking in model-based systems, Reiss [160] proposes a general system to manage consistency between software development artifacts. He suggests to inject information from all types of artifacts into a database and to establish consistency rules in the form of database queries and guards in order to ensure consistency. Similarly, Nentwich et al. [128] have provided the xlinkit framework, which can be used to keep artifacts in XML representation consistent. This framework also provides mechanisms to check consistency based on a set of user-definable rules expressed in XML.

The latter approaches address a relatively narrow field of trace usage. In the process of improving and tailoring traceability to the actual needs and goals of a project, future research should also aim to widen the immediate benefit to the recorders of traces in order to motivate them and to positively influence the human factor in traceability.

6 Future challenges

In the 1990s, traceability research has received the first broad attention in the requirements engineering community with the solid empirical groundwork of [74,155–157], the European NATURE project [94], and others. Now, about 15 years later, several advances have been accomplished as described in the previous sections. Nevertheless, there are still many open questions to be answered and issues to be solved in almost all areas of traceability. This section wraps up the open issues identified in the previous sections and complements them with additional issues reported in recent publications. These issues also overlap with the list of current challenges identified and detailed by the *Workshop on the Great Challenges of Traceability* (*GCT*) [37], and with a similar list assembled in the ECMDA traceability workshop [136]. The open issues are:

Agreeing on the foundations of traceability Until now, there is no agreement on a thoroughly investigated foundation for traceability. Particularly in the context of MDD, a common definition and understanding of traceability and its related concepts has to be found. Limón and Garbajosa [109] observe that *"[there] is a lack of a commonly accepted traceability definition further than the term definition"* and call for a unified traceability scheme including a link dataset, a link type set, a minimum set of links, and a set of metrics for traces. Most likely, there is no single universal traceability scheme, but a set of schemes with each one being suitable for a different domain or process. In either case, suitable traceability schemes and guidelines on how to adapt them to different settings have to be defined and evaluated.

Defining criteria for traceability metamodels At the very least, descriptive metrics for a traceability scheme have to be determined and aspects such as granularity, completeness, correctness, and quality of a single traceability link as well as of the whole set of traces recorded for a project has to be defined precisely. Then, the influence of varying occurrences of these characteristics on the traces has to be investigated.

Defining and using semantic aspects of traces Despite the need for different types of traceability links, which is driven by the intended usage of the traces [2], most automatic approaches to trace recording only support one general type of traceability link. Also there's a mismatch between the requirements traceability community, which has an understanding of semantics as meaning of a link (e.g., IS- IMPLEMENTED- BY) and the modeling community, in semantics is comprehended more in the concrete technical context of conditions, events, and actions. Merging both views could be beneficial for both communities. At large, there is also a trade-off between applying resource-extensive, but semantically more accurate manual techniques, and cost-efficient, but inaccurate automatic approaches. Bridging this gap, and finding a happy medium is one of the great challenges of traceability research.

Making research results comparable Current research results are often not very comparable. While Hayes and Dekhtyar [80] have presented a very useful framework to compare requirements traceability link retrieval algorithms, the framework has only been used by the publishing authors so far. Maybe one reason for this is that

the framework has not been designed for the comparison of complete traceability methods or tools in general, but only for evaluating and comparing automatic approaches to trace detection. In the future, a more extensive comparison scheme should be developed which includes aspects such as trace granularity and intended trace usage targeted by the approach. Ideally, such a scheme would also include a standard suite of reference artifacts and trace sets. This data could then be used to evaluate new approaches easily and consistently. Developing and establishing such a scheme in the research community could also improve the identification of future research opportunities.

Furthermore, the traceability research community itself is rather heterogeneous. In this article, we have concentrated on the requirements engineering and modeling communities, but there are more. This leads to varying perspectives on concepts and to different focuses. For example, most descriptions of tools or methods supporting traceability only concentrate on a specific usage scenario, such as change analysis and propagation, verification and validation, or system understanding. These different notions result in different subjective views of which quality and detail is desirable for traces. This, in turn, affects the evaluation of the experimental results. These differences are another issue that should be addressed by a future comparison framework in order to enable more objective evaluations.

Improving support for real-world problems Concerning traceability practices, a lot of effort has been put into addressing and overcoming technical limitations of traceability. However, the support of traceability research for practical problems in the industry is perceived as rather low:

> *[There] is a large gap between needs [...] in industry on one side and published solutions from academic research on the other.* [162].

Consequently, traceability research has to focus more on providing both practical solutions and support for applying them to real-world projects [27]. This can be achieved by providing training for traceability, by making traceability a relevant practical skill, and by focusing on a better technology transfer of successful traceability techniques to the industry. This can, however, not be achieved by researchers alone, but has to be supported by the industry as well. It is impossible to create representative research results without applying new technologies to industrial environments.

Methodically exploring the benefits of traceability In Sect. 3.2, we have described quite a lot of possible usage scenarios for traceability which we have derived from the literature. The evidence we have used, however, is merely of anecdotal nature. What is missing, is a thorough, methodical, and empirical exploration of the benefits of applying traceability techniques to software development processes in general. Significant empirical evidence of the benefits of traceability can also motivate the development of more usage-oriented traceability methods and solutions.

Support traceability as part of the development process Traceability is rarely directly supported by current software development processes [159]. There is a lack of guidance, both with respect to traceability planning and the evaluation of cost and benefit, and with respect to when and how actually carrying out traceability-related tasks during software development. In MDD, traceability should be regarded not only as an output of model transformation, but considered in context of the larger development process. This calls for addressing pre- and post-modeling traceability and also for traceability of transformation specifications.

In addition, industrial projects need guidance in order to determine goals, potential intentions and usage scenarios for traces. The appropriateness of different cost-efficient automatic traceability methods depends on the project's structure and artifacts. This has to be taken into consideration when developing strategy selection guidelines. Furthermore, the project's staff needs instructions on how to set up traceability procedures, structures, and schemes properly. Finally, support for selecting granularity, quality, and completeness of the traces to be captured is required.

Leveraging tool support In requirements engineering, tools have not changed much with respect to traceability in the past decade: Recording and using traces is to a large part time-consuming and tedious. In MDD, configuration of tools and transformations for traceability is neither standardized nor supported well. Both areas suffer from tool integration issues: Most research projects only cover the special case of traceability between two particular types of artifacts. In the industry, however, there is a need for more holistic approaches to traceability between artifacts created using different requirements management tools, modeling tools, integrated development environments (IDEs), and communication tools. If all of the tools along the development process would implement standardized interfaces, or at least would support a common trace data format, traces would become easier to record, manage and maintain. Hence, it is necessary to address these topics—not only from a researcher's perspective, but also in cooperation with tool vendors in order to achieve widespread tool support.

Addressing heterogeneous environments Besides the integration of tools, traceability methods should also be more suited to industrial projects, which often have to deal with documents which are large and only weakly structured [35]. In contrast to this, current approaches often build on

information sources which are homogeneous and more or less structured. Thus, support for real-world environments has to be enhanced. Also, it has to be acknowledged that software development artifacts exist in very different forms of representation, and that prior to recording fine-grained traceability links, it should be recorded how artifacts themselves are created, evolved and transformed at a macro level [75].

Capturing informal aspects There is more information relevant to traceability than just models and artifacts stored in different tools. Discussions, alternatives, and rationale are often present in oral conversations but they get lost, if nobody writes them down. In these areas of intuitive and informal communication, a non-intrusive and lightweight method is needed to capture the valuable information without hindering the communication and discussion process itself [24]. In addition, the captured information has to be included in the trace repository in a manageable way.

Enabling global, distributed traceability If the field of observation is extended from internal software processes to end-to-end traceability in global and distributed development processes involving several companies and teams, there is much work left to be done in order to integrate these potentially heterogeneous environments. Distributed traceability is in most parts an unexplored area. The same is true for the reuse of traces between projects—for example when pieces of domain knowledge or requirements from earlier projects are reused.

Improving usability of traceability Query, navigation, and visualization techniques have not been investigated in much detail, either. As with traceability planning, a concise catalog, describing which techniques are useful for which usage scenarios would be helpful. Arkley and Riddle [16] have shown that users are willing to invest some extra effort into trace recording if they are rewarded with some benefit. Applying usability practices and ergonomics to trace query, navigation, presentation, and particularly visualization, could potentially raise the acceptance among the users. At the same time, scalable visualization techniques can help to manage the huge amounts of traces occurring in real-world projects.

Supporting maintenance of traces Maintenance of traces is another area in which open issues have to be investigated. As requirements, models, and software tend to change constantly during the development process, traces have to be kept up to date. In particular, dependencies which are introduced unknowingly have to be addressed. In the past, changing dependencies have mainly been handled by creating new traceability links, and by deleting outdated ones, where needed. However, doing so results in lost traces and thus, in lost information. To avoid this, traces have to be versioned. This has been addressed in some special areas (e.g., by Nguyen et al. [131] for hypermedia), but so far, there is no holistic approach addressing the versioning of traces and exploring its consequences.

In summary, in order to provide useful solutions to the industry, traceability costs have to be minimized. This is very well appreciated by current researchers. However, in order to mature traceability, the research community has to improve its basic understanding and explicitly provide benefit both to project leaders and to the recorders of traceability. To achieve the latter, research has to focus on aspects such as tailoring, customization, usability, and ergonomics. Furthermore, situational best practices, guidelines, and case studies have to be provided and made known in the industrial area.

7 Conclusion

Traceability has been around for almost 30 years now—the first requirements traceability tool has been presented in 1978 [143]. Traceability research has gained increasing attention and importance since its first thorough investigation in the 1990s. Since 2002, there have been regular workshops covering traceability—most prominently the series of *Traceability in Emerging Forms of Software Engineering* (*TEFSE*) for the requirements traceability community and the *Traceability Workshop of the European Conference on Model-Driven Architecture* (*ECMDA-TW*). This development will most likely continue in the future and traceability will continuously gain in importance.

This article has surveyed and covered several aspects of traceability research and practice both in the requirements engineering and the modeling communities. During our analysis we have made several interesting observations:

First of all, and most obviously, the history of requirements traceability is longer than that of traceability in MDD. While the former can be measured in decades, the latter would still be measured in years. This is why most basic topics, such as definitions of terms, metamodels and link type classifications, are still not agreed upon in the MDD community. However, due to the basic notion of models—expressed in the form of standardized meta-metamodels—the basic formalisms, elements, and structures are already implicitly present, which is an advantage over requirements traceability. The latter seems to be more mature—but only in part: while the definition of traceability by Gotel and Finkelstein [74] is commonly accepted and cited as the standard definition, there are still no standardized traceability schemes, metamodels, and link type classifications.

Considering the scope of traceability, requirements traceability incorporates a more holistic view: Because of the general goal to follow the whole life of requirements from its sources to the products generated from them, basically the

Table 3 Summary of limitations, general approaches, and future challenges

Topic	General approaches/solutions	Important challenges
Basics of traceability	Many single proposals of concepts, classifications, types, etc.	Standardization based on common, accepted, and sound foundation
Trace integration	Centralized trace repository	Standardization and adoption by tools
	Tool interoperation through standardized interfaces	Seamless integration into development process/activities
Trace recording	Derive links by applying rules to artifacts and existing traces	Generalize approaches and implement them in standard tools
	Derive links by logging and analyzing user activities	Provide recipes of how to achieve what
	Derive links by analyzing the artifacts content (i.e., language)	Make them easy and enjoyable to use (see human factors)
Trace maintenance	Approaches generally based on some form of events, triggers, or rules	Address implicit evolution
		Support management of versioned traces
Economic aspects	Single reports of traceability measures tailored to the specific needs of a project, company, etc.	Research cost and benefit of traceability
		Develop general guidelines and best practices based on sound research
Human aspects	Single reports of usability by personal benefit	Generalize observations and move from particular solutions to global usability implemented in tools

whole software development process is addressed. In contrast to this, MDD focuses more on traceability of model elements through consecutive transformation steps. Therefore, requirements traceability is hit by the problem of recording traceability links between mostly informal, weakly structured, and diverse artifacts, and integrating these links. Instead, in MDD, traceability links can be produced quite easily by just enhancing transformations or using transformation tools with built-in traceability support. But the rather narrow focus of MDD also leads to problems: On the one hand, traceability links bridging the gap between non-models and models have to be recorded and expressed somehow. On the other hand, several aspects (e.g., high-level goals) of traceability depend on external factors which are out of scope of MDD themselves. This again leads to a limited motivation for advanced, more global approaches to traceability in MDD research.

Beyond these differences, there are also common—and challenging—problems: Both fields give several examples of how traces can be used to achieve goals. However, these examples are almost always considered in isolation. We are missing a framework that can be used to determine the type and granularity of traceability links needed in order to achieve a set of given goals. Such a framework should also include recipes which describe how these links have to be used to achieve the respective goals.

In addition, traceability methods are not used in practice as much as they could. One of the main reasons is lack of good tool support. In requirements traceability this includes recording, integrating, and using traceability links. In MDD, traceability is often not supported natively by transformation tools, but it is easy to enhance transformations and to easily record just the needed traceability links as by-products of transformations. Storing links is also not a problem due to standard model serialization formats. However, when it comes to using—and particularly visualizing—traceability links, current tools provide no support at all, and consequently, traceability links can often not be put to use.

In Sects. 5 and 6 we have analyzed extensively recent research and future challenges in several areas of traceability. Table 3 summarizes our findings by naming the most significant approaches or solutions achieved in these areas and the most important open issues, respectively.

To conclude, in order to leverage traceability research in general, both fields should be integrated to some extent. The standardized basis and the automated recording opportunities of MDD in combination with a more holistic view of requirements traceability provide a good common basis for this. A current problem is that traceability research is done in a lot of different sub-communities. There is no common forum for discussing traceability across the different fields of research. A common workshop involving at least the communities of RE/TEFSE and MODELS/ECMDA-TW could be a first start into a regular exchange of information and an opportunity to reach a more common view of traceability.

References

1. Aizenbud-Reshef, N., Nolan, B.T., Rubin, J., Shaham-Gafni, Y.: Model traceability. IBM Syst. J. **45**(3), 515–526 (2006)
2. Aizenbud-Reshef, N., Paige, R.F., Rubin, J., Shaham-Gafni, Y., Kolovos, D.S.: Operational semantics for traceability. In: ECMDA Traceability Workshop (ECMDA-TW) 2005 Proceedings, pp. 8–14. Sintef, Trondheim (2005). ISBN 978-82-14-03813-2

3. Albinet, A., Boulanger, J.L., Dubois, H., Peraldi-Frati, M.A., Sorel, Y., Van, Q.D.: Model-based methodology for requirements traceability in embedded systems. In: ECMDA Traceability Workshop (ECMDA-TW) 2007 Proceedings, pp. 27–36. Sintef, Trondheim (2007). ISBN 978-82-14-04056-2
4. Alexander, I.: Towards automatic traceability in industrial practice. In: Proceedings of the 1st International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE '02). ACM, New York (2002)
5. Alexander, I.: Semiautomatic tracing of requirement versions to use cases. In: Proceedings of the 2nd International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE '03). ACM, New York (2003)
6. Alexander, I.: A taxonomy of stakeholders, human roles in system development. In: Stahl, B.C. (ed.) Issues and Trends in Technology and Human Interaction, pp. 25–71. IRM Press, Hershey (2006). ISBN 978-1-599-04269-5
7. Alexander, I., Robertson, S., Maiden, N.: What influences the requirements process in industry? A report on industrial practice. In: 13th IEEE International Requirements Engineering Conference (RE'05) Proceedings, pp. 411–415. IEEE Computer Society, New York (2005)
8. Amar, B., Leblanc, H., Coulette, B.: A traceability engine dedicated to model transformation for software engineering. In: ECMDA Traceability Workshop (ECMDA-TW) 2008 Proceedings, pp. 7–16. Sintef, Trondheim (2008). ISBN 978-82-14-04396-9
9. Anderson, K.M., Taylor, R.N., Whitehead, E.J. Jr.: Chimera: hypermedia for heterogeneous software development environments. ACM Trans. Inf. Syst. **18**(3), 211–245 (2000)
10. Anquetil, N., Grammel, B., Galvão, I., Noppen, J., Khan, S.S., Arboleda, H., Rashid, A., Garcia, A.: Traceability for model driven, software product line engineering. In: ECMDA Traceability Workshop (ECMDA-TW) 2008 Proceedings, pp. 77–86. Sintef, Trondheim (2008). ISBN 978-82-14-04396-9
11. ANSI. ANSI-X3.138-1988: Information Resource Dictionary System (IRDS). American National Standards for Information Systems, New York (1988)
12. Antoniol, G., Canfora, G., Casazza, G., Lucia, A.D.: Maintaining traceability links during object-oriented software evolution. Softw. Pract. Experience **31**(4), 331–355 (2001)
13. Antoniol, G., Canfora, G., Casazza, G., Lucia, A.D., Merlo, E.: Recovering traceability links between code and documentation. IEEE Trans. Softw. Eng. **28**(10), 970–983 (2002)
14. Antoniol, G., Merlo, E., Guéhéneuc, Y.G., Sahraoui, H.: On feature traceability in object oriented programs. In: TEFSE '05: Proceedings of the 3rd International Workshop on Traceability in Emerging Forms of Software Engineering, pp. 73–78. ACM, New York (2005)
15. Arkley, P., Manson, P., Riddle, S.: Enabling traceability. In: Proceedings of the 1st International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE '02). ACM, New York (2002)
16. Arkley, P., Riddle, S.: Overcoming the traceability benefit problem. In: 13th IEEE International Conference on Requirements Engineering (RE'05) Proceedings, pp. 385–389. IEEE Computer Society, New York (2005)
17. Asuncion, H.U., François, F., Taylor, R.N.: An end-to-end industrial software traceability tool. In: ESEC-FSE '07: Proceedings of the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering, pp. 115–124. ACM, New York (2007)
18. Baeza-Yates, R.A., Ribeiro-Neto, B.: Modern Information Retrieval, 1st edn. Addison-Wesley, Boston (1999). ISBN 978-0-201-39829-8
19. Barbero, M., del Fabro, M.D., Bézivin, J.: Traceability and provenance issues in global model management. In: ECMDA Traceability Workshop (ECMDA-TW) 2007 Proceedings, pp. 47–55. Sintef, Trondheim (2007). ISBN 978-82-14-04056-2
20. Bézivin, J., Jouault, F., Rosenthal, P., Valduriez, P.: Modeling in the large and modeling in the small. In: Aßmann, U., Aksit, M., Rensink, A. (eds.) Model Driven Architecture: MDAFA 2003 and MDAFA 2004 Selected Papers. Lecture Notes in Computer Science, vol. 3599, pp. 33–46. Springer, Berlin (2005). ISBN 978-3-540-28240-2
21. Blanc, X., Gervais, M.P., Sriplakich, P.: Model bus: towards the interoperability of modelling tools. In: Aßmann, U., Aksit, M., Rensink, A. (eds.) Model Driven Architecture: MDAFA 2003 and MDAFA 2004 Selected Papers. Lecture Notes in Computer Science, vol. 3599, pp. 17–32. Springer, Berlin (2005). ISBN 978-3-540-28240-2
22. Boldyreff, C., Nutter, D., Rank, S.: Active artefact management for distributed software engineering. In: Proceedings of the 26th IEEE Annual International Conference on Computer Software and Applications, pp. 1081–1086. IEEE Computer Society, New York (2002)
23. Brasethvik, T., Gulla, J.A.: Natural language analysis for semantic document modeling. Data Knowl. Eng. **38**(1), 45–62 (2001)
24. Braun, A., Bruegge, B., Dutoit, A.: Supporting informal meetings in requirements engineering. In: Requirements Engineering: Foundation for Software Quality (REFSQ'01) Proceedings, pp. 26–40. http://refsq.org (2001)
25. Brcina, R., Riebisch, M.: Defining a traceability link semantics for design decision support. In: ECMDA Traceability Workshop (ECMDA-TW) 2008 Proceedings, pp. 39–48. Sintef, Trondheim (2008). ISBN 978-82-14-04396-9
26. Briand, L.C., Labiche, Y., O'Sullivan, L.: Impact analysis and change management of UML models. In: ICSM '03: Proceedings of the International Conference on Software Maintenance, pp. 256–265. IEEE Computer Society, New York (2003)
27. Brinkkemper, S.: Requirements engineering research the industry is and is not waiting for. In: Requirements Engineering: Foundation for Software Quality (REFSQ'04) Proceedings, pp. 41–54. http://refsq.org (2004)
28. Brooks, F.P.: No silver bullet: essence and accidents of software engineering. IEEE Comp. **20**(4), 10–19 (1987)
29. Bui, T.X., Bodart, F., Ma, P.C.: ARBAS: a formal language to support argumentation in network-based organizations. J. Manag. Inf. Syst. **14**(3), 223–237 (1997). ISSN 0742-1222
30. Campos, P., Nunes, N.J.: Practitioner tools and workstyles for user-interface design. IEEE Softw. **24**(1), 73–80 (2007)
31. Card, D.N.: Designing software for producibility. J. Syst. Softw. **17**(3), 219–225 (1992)
32. Cerbah, F., Euzenat, J.: Traceability between models and texts through terminology. Data Knowl. Eng. **38**(1), 31–43 (2001)
33. Chen, J.Y.J., Chou, S.C.: Consistency management in a process environment. J. Syst. Softw. **47**, 105–110 (1999)
34. Chikofsky, E.J., Rubenstein, B.L.: CASE: reliability engineering for information systems. IEEE Softw. **5**(2), 11–16 (1988)
35. Cleland-Huang, J., Berenbach, B., Clark, S., Settimi, R., Romanova, E.: Best practices for automated traceability. IEEE Comp. **40**(6), 27–35 (2007)
36. Cleland-Huang, J., Chang, C.K., Christensen, M.: Event-based traceability for managing evolutionary change. IEEE Trans. Softw. Eng. **29**(9), 796–810 (2003)
37. Cleland-Huang, J., Dekhtyar, A., Hayes, J.H.: Problem Statements and Grand Challenges. Tech. Rep. COET-GCT-06-01-0.9, Center of Excellence for Traceability. http://www.traceabilitycenter.org/downloads/documents/GrandChallenges/ (2006)
38. Cleland-Huang, J., Habrat, R.: Visual support in automated tracing. In: 2nd International Workshop on Requirements Engineering

Visualization (REV 2007). IEEE Computer Society, New York (2007)
39. Cleland-Huang, J., Chang, K.C., Wise, J.C.: Automating performance related impact analysis through event based traceability. Requir. Eng. J. **8**(3), 171–182 (2003)
40. Cleland-Huang, J., Settimi, R., Duan, C., Zou, X.: Utilizing supporting evidence to improve dynamic requirements traceability. In: 13th IEEE International Requirements Engineering Conference (RE'05) Proceedings, pp. 135–144. IEEE Computer Society, New York (2005)
41. Cleland-Huang, J., Zemont, G., Lukasik, W.: A heterogeneous solution for improving the return on investment of requirements traceability. In: 12th IEEE International Requirements Engineering Conference (RE'04) Proceedings, pp. 230–239. IEEE Computer Society, New York (2004)
42. Conklin, J., Begeman, M.L.: gIBIS: a hypertext tool for exploratory policy discussion. ACM Trans. Inf. Syst. **6**(4), 303–331 (1988)
43. Costa, M., da Silva, A.R.: RT-MDD framework—a practical approach. In: ECMDA Traceability Workshop (ECMDA-TW) 2007 Proceedings, pp. 17–26. Sintef, Trondheim (2007). ISBN 978-82-14-04056-2
44. Dahlstedt, Å.G., Persson, A.: Requirements interdependencies: state of the art and future challenges. In: Engineering and Managing Software Requirements, pp. 95–116. Springer, Berlin (2005). ISBN 978-3-540-25043-2
45. de Lucia, A., Fasano, F., Oliveto, R., Tortora, G.: ADAMS Re-Trace: a traceability recovery tool. In: CSMR '05: Proceedings of the 9th European Conference on Software Maintenance and Reengineering, pp. 32–41. IEEE Computer Society, New York (2005)
46. de Lucia, A., Fasano, F., Oliveto, R., Tortora, G.: Recovering traceability links in software artefact management systems using information retrieval methods. ACM Trans. Softw. Eng. Meth. **16**(4), 13/1–13/50 (2007)
47. Decker, B., Ras, E., Rech, J., Klein, B., Höcht, C.: Self-organized reuse of software engineering knowledge supported by semantic wikis. In: Workshop on Semantic Web Enabled Software Engineering (SWESE) Proceedings. http://www.mel.nist.gov/msid/conferences/SWESE/ (2005)
48. Dekhtyar, A., Hayes, J.H., Sundaram, S., Holbrook, A., Dekhtyar, O.: Technique integration for requirements assessment. In: 15th IEEE International Requirements Engineering Conference (RE'07) Proceedings, pp. 141–150. IEEE Computer Society, New York (2007)
49. Deng, M., Stirewalt, R.E.K., Cheng, B.H.C.: Retrieval by construction: a traceability technique to support verification and validation of UML formalizations. Int. J. Softw. Eng. Knowl. Eng. **15**(5), 837–872 (2005)
50. Derniame, J.C., Kaba, B.A., Wastell, D.G. (eds.): Software Process: Principles, Methodology, Technology. Lecture Notes in Computer Science, vol. 1500. Springer, Berlin (1999). ISBN 978-3-540-65516-9
51. Dick, J.: Rich traceability. In: Proceedings of the 1st International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE '02). ACM, New York (2002)
52. Dömges, R., Pohl, K.: Adapting traceability environments to project-specific needs. Commun. ACM **41**(12), 54–62 (1998)
53. dos Santos Soares, M., Vrancken, J.L.M.: Model-driven user requirements specification using SysML. J. Softw. **3**(6), 57–68 (2008)
54. Drivalos, N., Kolovos, D.S., Paige, R.F., Fernandes, K.J.: Engineering a DSL for software traceability. In: Gaševic, D., Lämmel, R., van Wyk, E. (eds) Proceedings of the 1st International Conference on Software Languages Engineering, SLE '08. Lecture Notes in Computer Science, vol. 5452. Springer, Berlin (2008). ISBN 978-3-642-00433-9
55. Duan, C., Cleland-Huang, J.: Visualization and analysis in automated trace retrieval. In: 1st International Workshop on Requirements Engineering Visualization (REV'06). IEEE Computer Society, New York (2006)
56. Dutoit, A.H., McCall, R., Mistrik, I., Paech, B. (eds.): Rationale Management in Software Engineering. Springer, Berlin (2006). ISBN 978-3-540-30997-0
57. Dutoit, A.H., Paech, B.: Rationale management in software engineering. In: Chang, S.K. (ed.) Handbook of Software Engineering and Knowledge Engineering, vol. 1—Fundamentals, pp. 787–816. World Scientific, Singapore (2001). ISBN 978-9-810-24973-1
58. Ebner, G., Kaindl, H.: Tracing all around in reengineering. IEEE Softw. **19**(3), 70–77 (2002)
59. Egyed, A.: Tailoring software traceability to value-based needs. In: Biffl, S., Aurum, A., Boehm, B., Erdogmus, H., Grünbacher, P. (eds.) Value-Based Software Engineering. Springer, Berlin (2005). ISBN 978-3-540-25993-0
60. Egyed, A.: Fixing inconsistencies in UML design models. In ICSE '07: Proceedings of the 29th International Conference on Software Engineering pp. 292–301. IEEE Computer Society, New York (2007)
61. Egyed, A., Grünbacher, P.: Supporting software understanding with automated requirements traceability. Int. J. Softw. Eng. Knowl. Eng. **15**(5), 783–810 (2005)
62. Egyed, A., Grünbacher, P., Heindl, M., Biffl, S.: Value-based requirements traceability: lessons learned. In: 15th IEEE International Requirements Engineering Conference (RE'07) Proceedings, pp. 115–118. IEEE Computer Society, New York (2007)
63. Espinoza, A., Alarcon, P.P., Garbajosa, J.: Analyzing and systematizing current traceability schemas. In: Proceedings of the 30th Annual IEEE/NASA Software Engineering Workshop, pp. 21–32. IEEE Computer Society, New York (2006)
64. Falleri, J.R., Huchard, M., Nebut, C.: Towards a traceability framework for model transformations in Kermeta. In: ECMDA Traceability Workshop (ECMDA-TW) 2006 Proceedings, pp. 31–40. Sintef, Trondheim (2006). ISBN 978-82-14-04030-2
65. France, R., Rumpe, B.: Model-driven development of complex software: a research roadmap. In: FOSE '07: Future of Software Engineering Proceedings, pp. 37–54. IEEE Computer Society, New York (2007)
66. Fritzsche, M., Johannes, J., Zschaler, S., Zherebtsov, A., Terekhov, A.: Application of tracing techniques in model-driven performance engineering. In: ECMDA Traceability Workshop (ECMDA-TW) 2008 Proceedings, pp. 111–120. Sintef, Trondheim (2008). ISBN 978-82-14-04396-9
67. Galvão, I., Göknil, A.: Survey of traceability approaches in model-driven engineering. In: Proceedings of the 11th IEEE International EDOC Enterprise Computing Conference, pp. 313–324. IEEE Computer Society, New York (2007)
68. Gervasi, V., Zowghi, D.: Reasoning about inconsistencies in natural language requirements. ACM Trans. Softw. Eng. Meth. **14**(3), 277–330 (2005)
69. Gills, M.: Survey of traceability models in IT projects. In: ECMDA Traceability Workshop (ECMDA-TW) 2005 Proceedings, pp. 39–46. Sintef, Trondheim (2005). ISBN 978-82-14-03813-2
70. Glitia, F., Etien, A., Dumoulin, C.: Fine grained traceability for an MDE approach of embedded system conception. In: Oldevik, J. Aagedal, J.Ø. (eds.) ECMDA Traceability Workshop (ECMDA-TW) 2008 Proceedings, pp. 27–38. Sintef, Trondheim (2008). ISBN 978-82-14-04396-9
71. Goguen, J.A.: Formality and informality in requirements engineering. In: 2nd International Requirements Engineering

Conference (ICRE'96) Proceedings. IEEE Computer Society, New York (1996)
72. Göknil, A., Kurtev, I., van den Berg, K.: Change impact analysis based on formalization of trace relations for requirements. In: ECMDA Traceability Workshop (ECMDA-TW) 2008 Proceedings, pp. 59–75. Sintef, Trondheim (2008). ISBN 978-82-14-04396-9
73. Gotel, O.C.Z.: Contribution Structures for Requirements Traceability. Ph.D. thesis, Imperial College of Science, Technology and Medicine, University of London (1995)
74. Gotel, O.C.Z., Finkelstein, A.C.W.: An analysis of the requirements traceability problem. In: 1st IEEE International Requirements Engineering Conference (RE'94) Proceedings, pp. 94–101. IEEE Computer Society, New York (1994)
75. Gotel, O.C.Z., Morris, S.J.: Macro-level traceability via media transformations. In: Paech, B., Rolland, C. (eds.) Requirements Engineering: Foundation for Software Quality (REFSQ'08) Proceedings. Lecture Notes in Computer Science, vol. 5025, pp. 129–134. Springer, Berlin (2008). ISBN 978-3-540-69060-3
76. Grechanik, M., McKinley, K.S., Perry, D.E.: Recovering and using use-case-diagram-to-source-code traceability links. In: ESEC-FSE '07: Proceedings of the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering, pp. 95–104. ACM, New York (2007)
77. Hapke, M., Jaszkiewicz, A., Kowalczykiewicz, K., Weiss, D., Zielniewicz, P.: OPHELIA—open platform for distributed software development. In: Open Source for an Information and knowledge society: Proceedings of the Open Source International Conference. Malaga, Spain. http://malaga04.opensourceworldconference.com/ (2004)
78. Hardman, L., Bulterman, D.C.A., van Rossum, G.: The Amsterdam hypermedia model: adding time and context to the Dexter model. Commun. ACM **37**(2), 50–62 (1994)
79. Haumer, P., Pohl, K., Weidenhaupt, K., Jarke, M.: Improving reviews by extended traceability. In: Proceedings of the 32nd Hawaii International Conference on System Sciences. IEEE Computer Society, New York (1999)
80. Hayes, J.H., Dekhtyar, A.: A framework for comparing requirements tracing experiments. Int. J. Softw. Eng. Knowl. Eng. **15**(5), 751–781 (2005)
81. Hayes, J.H., Dekhtyar, A.: Humans in the traceability loop: can't live with 'em, can't live without 'em. In: TEFSE '05: Proceedings of the 3rd International Workshop on Traceability in Emerging Forms of Software Engineering, pp. 20–23. ACM, New York (2005)
82. Hayes, J.H., Dekhtyar, A., Sundaram, S.K.: Advancing candidate link generation for requirements tracing: the study of methods. IEEE Trans. Softw. Eng. **32**(01), 4–19 (2006)
83. Hayes, J.H., Dekhtyar, A., Sundaram, S.K., Howard, S.: Helping analysts trace requirements: an objective look. In: 12th IEEE International Requirements Engineering Conference (RE'04) Proceedings, pp. 249–259. IEEE Computer Society, New York (2004)
84. Heindl, M., Biffl, S.: A case study on value-based requirements tracing. In: Proceedings of the Joint 10th European Software Engineering Conference (ESEC) and the 13th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE-13), pp. 60–69. ACM, New York (2005)
85. Heindl, M., Biffl, S.: Modeling of requirements tracing. In: Meyer, B., Nawrocki, J.R., Walter, B. (eds.) Balancing Agility and Formalism in Software Engineering (CEE-SET 2007). Lecture Notes in Computer Science, vol. 5082, pp. 267–278. Springer, Berlin (2008)
86. Herman, I., Melancon, G., Marshall, M.S.: Graph visualization and navigation in information visualization: a survey. IEEE Trans. Vis. Comp. Graph. **06**(1), 24–43 (2000)
87. Hoffmann, M., Kühn, N., Bittner, M.: Requirements for requirements management tools. In: 12th IEEE International Conference on Requirements Engineering (RE'04) Proceedings, pp. 301–308. IEEE Computer Society, New York (2004)
88. Hofmann, H.F., Lehner, F.: Requirements engineering as a success factor in software projects. IEEE Softw. **18**(4), 58–66 (2001)
89. IEEE: IEEE Guide to Software Requirements Specification, ANSI/IEEE Std 830-1984. IEEE Press, Piscataway (1984)
90. IEEE: IEEE Standard Glossary of Software Engineering Terminology. IEEE Press, Piscataway (1990)
91. Ilieva, M.G., Ormandjieva, O.: Models derived from automatically analyzed textual user requirements. In: 4th International Conference on Software Engineering Research, Management and Applications, pp. 13–21. IEEE Computer Society, New York (2006)
92. Ilieva, M., Ormandjieva, O.: Automatic transition of natural language software requirements specification into formal presentation. In: Montoyo, A., Muñoz, R., Métais, E. (eds.) Natural Language Processing and Information Systems. Lecture Notes in Computer Science, vol. 3513, pp. 392–397. Springer, Berlin (2005)
93. Jacobson, I.: Object-oriented development in an industrial environment. In: OOPSLA '87: Conference Proceedings on Object-Oriented Programming Systems, Languages and Applications, pp. 183–191. ACM, New York (1987)
94. Jarke, M., Rolland, C., Sutcliffe, A. (eds.): The NATURE of Requirements Engineering. Shaker, Aachen (1999). ISBN 978-3-8265-6174-0
95. Jarke, M.: Requirements tracing. Commun. ACM **41**(12), 32–36 (1998)
96. Jarke, M., Pohl, K.: Information systems quality and quality informations systems. In: The Impact of Computer Supported Technologies in Information Systems Development, IFIP Transactions, pp. 345–375. Elsevier Science, North-Holland (1992)
97. Jirapanthong, W., Zisman, A.: XTraQue: traceability for product line systems. J. Softw. Syst. Model. **8**(1), 117–144 (2009)
98. Jouault, F.: Loosely coupled traceability for ATL. In: ECMDA Traceability Workshop (ECMDA-TW) 2005 Proceedings, pp. 29–37. Sintef, Trondheim (2005). ISBN 978-82-14-03813-2
99. Kaindl, H., Kramer, S., Diallo, P.S.N.: Semiautomatic generation of glossary links: a practical solution. In: HYPERTEXT '99: Proceedings of the Tenth ACM Conference on Hypertext and Hypermedia: Returning to our Diverse Roots, pp. 3–12. ACM, New York (1999)
100. Kirova, V., Kirby, N., Kothari, D., Childress, G.: Effective requirements traceability: models, tools, and practices. Bell Labs Tech. J. **12**(4), 143–157 (2008)
101. Knuth, D.E.: Literate programming. Comp. J. **27**(2), 97–111 (1984)
102. Kolovos, D.S., Paige, R.F., Polack, F.A.C.: On-demand merging of traceability links with models. In: ECMDA Traceability Workshop (ECMDA-TW) 2006 Proceedings, pp. 6–14. Sintef, Trondheim (2006). ISBN 978-82-14-04030-2
103. Königs, A., Schürr, A.: MDI—a rule-based multi-document and tool integration approach. J. Softw. Syst. Model. **5**(4), 349–368 (2006)
104. Kwan, I., Damian, D., Storey, M.A.: Visualizing a requirements-centred social network to maintain awareness within development teams. In: 1st International Workshop on Requirements Engineering Visualization (REV'06). IEEE Computer Society, New York (2006)
105. Lago, P., Muccini, H., van Vliet, H.: A scoped approach to traceability management. J. Syst. Softw. **82**(1), 168–182 (2009)
106. Lago, P., Niemelä, E., van Vliet, H.: Tool support for traceable product evolution. In: CSMR '04: Proceedings of the 8th Euro-

micro Working Conference on Software Maintenance and Reengineering, pp. 261–269. IEEE Computer Society, New York (2004)
107. Lee, C., Guadagno, L., Jia, X.: An agile approach to capturing requirements and traceability. In: Proceedings of the 2nd International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE '03). ACM, New York (2003)
108. Letelier, P.: A framework for requirements traceability in UML-based projects. In: Proceedings of the 1st International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE '02), pp. 32–41. ACM, New York (2002)
109. Limón, A.E., Garbajosa, J.: The need for a unifying traceability scheme. In: ECMDA Traceability Workshop (ECMDA-TW) 2005 Proceedings, pp. 47–56. Sintef, Trondheim (2005). ISBN 978-82-14-03813-2
110. Lin, J., Lin, C.C., Cleland-Huang, J., Settimi, R., Amaya, J., Bedford, G., Berenbach, B., Khadra, O.B., Duan, C., Zou, X.: Poirot: a distributed tool supporting enterprise-wide traceability. In: 14th IEEE International Requirements Engineering Conference (RE'06) Proceedings, pp. 363–364. IEEE Computer Society, New York (2006)
111. Lindvall, M.: A study of traceability in object-oriented systems development. Ph.D. thesis, Department of Computer and Information Science, Linköping University (1994)
112. Lormans, M., van Deursen, A.: Reconstructing requirements coverage views from design and test using traceability recovery via LSI. In: TEFSE '05: Proceedings of the 3rd International Workshop on Traceability in Emerging Forms of Software Engineering, pp. 37–42. ACM, New York (2005)
113. Lormans, M., van Deursen, A.: Can LSI help reconstructing requirements traceability in design and test? In: CSMR '06: Proceedings of the Conference on Software Maintenance and Reengineering, pp. 47–56. IEEE Computer Society, New York (2006)
114. Mäder, P., Gotel, O., Philippow, I.: Rule-based maintenance of post-requirements traceability relations. In: 16th IEEE International Requirements Engineering Conference (RE'08) Proceedings, pp. 23–32. IEEE Computer Society, New York (2008)
115. Mäder, P., Philippow, I., Riebisch, M.: Customizing traceability links for the unified process. In: Overhage, S., Szyperski, C.A., Reussner, R., Stafford, J.A. (eds.) Software Architectures, Components, and Applications (QoSA'07). Lecture Notes in Computer Science, vol. 4880. Springer, Berlin (2008)
116. Maletic, J.I., Collard, M.L., Simoes, B.: An XML based approach to support the evolution of model-to-model traceability links. In: TEFSE '05: Proceedings of the 3rd International Workshop on Traceability in Emerging Forms of Software Engineering, pp. 67–72. ACM, New York (2005)
117. Marcus, A., Maletic, J.I., Sergeyev, A.: Recovery of traceability links between software documentation and source code. Int. J. Softw. Eng. Knowl. Eng. **15**(4), 811–836 (2005)
118. Marcus, A., Xie, X., Poshyvanyk, D.: When and how to visualize traceability links? In: TEFSE '05: Proceedings of the 3rd International Workshop on Traceability in Emerging Forms of Software Engineering, pp. 56–61. ACM, New York (2005)
119. Mohan, K., Ramesh, B.: Managing variability with traceability in product and service families. In: Proceedings of the 35th Hawaii International Conference on System Sciences, pp. 76–94. IEEE Computer Society, New York (2002)
120. Mohan, K., Ramesh, B.: Traceability-based knowledge integration in group decision and negotiation activities. Decis. Support Syst. **43**(3), 968–989 (2007)
121. Mohan, K., Xu, P., Ramesh, B.: Supporting dynamic group decision and negotiation processes: a traceability augmented peer-to-peer network approach. Inf. Manag. **43**(5), 650–662 (2006)
122. Moran, T.P., Carroll, J.M. (eds.): Design Rationale: Concepts, Techniques, and Use. Lawrence Erlbaum, Hillsdale (1995). ISBN 978-0-805-81567-2
123. Munson, E.V., Nguyen, T.N.: Concordance, conformance, versions, and traceability. In: TEFSE '05: Proceedings of the 3rd International Workshop on Traceability in Emerging Forms of Software Engineering, pp. 62–66. ACM, New York (2005)
124. Murta, L.G.P., van der Hoek, A., Werner, C.M.L.: Continuous and automated evolution of architecture-to-implementation traceability links. Autom. Softw. Eng. **15**(1), 75–107 (2008)
125. Natt och Dag, J., Gervasi, V., Brinkkemper, S., Regnell, B.: A linguistic-engineering approach to large-scale requirements management. IEEE Softw. **22**(1), 32–39 (2005)
126. Natt och Dag, J., Regnell, B., Carlshamre, P., Andersson, M., Karlsson, J.: A feasibility study of automated natural language requirements analysis in market-driven development. Requir. Eng. J. **7**(1), 20–33 (2002)
127. Naslavsky, L., Alspaugh, T.A., Richardson, D.J., Ziv, H.: Using scenarios to support traceability. In: TEFSE '05: Proceedings of the 3rd International Workshop on Traceability in Emerging Forms of Software Engineering, pp. 25–30. ACM, New York (2005)
128. Nentwich, C., Emmerich, W., Finkelstein, A., Ellmer, E.: Flexible consistency checking. ACM Trans. Softw. Eng. Meth. **12**(1), 28–63 (2003)
129. Neumuller, C., Grunbacher, P.: Automating software traceability in very small companies: a case study and lessons learne. In: ASE '06: Proceedings of the 21st IEEE/ACM International Conference on Automated Software Engineering, pp. 145–156. IEEE Computer Society, New York (2006)
130. Nguyen, T.N., Munson, E.V.: The software concordance: a new software document management environment. In: SIGDOC '03: Proceedings of the 21st Annual International Conference on Documentation, pp. 198–205. ACM, New York (2003)
131. Nguyen, T.N., Thao, C., Munson, E.V.: On product versioning for hypertexts. In: SCM '05: Proceedings of the 12th International Workshop on Software Configuration Management, pp. 113–132. ACM, New York (2005)
132. Object Management Group: A Proposal for an MDA Foundation Model. Object Management Group, Needham, ormsc/05-04-01 ed. (2005)
133. Object Management Group: The Meta-Object Facility 2.0 Query/View/Transformation Specification. Final Adopted Specifiaction (2005)
134. Object Management Group: The Meta-Object Facility 2.0 Core Specification. Final Adopted Specifiaction (2006)
135. Object Management Group: OMG Systems Modeling Language. Version 1.1 (2008)
136. Oldevik, J., Aagedal, J.: Future Research Topics Discussion. ECMDA Traceability Workshop (ECMDA-TW '05). http://www.sintef.no/upload/10558/Future-Research-Topics.pdf (2005)
137. Oldevik, J., Neple, T.: Traceability in model to text transformations. In: ECMDA Traceability Workshop (ECMDA-TW) 2006 Proceedings, pp. 64–69. Sintef, Trondheim (2006). ISBN 978-82-14-04030-2
138. Olsen, G.K., Oldevik, J.: Scenarios of traceability in model to text transformations. In: Akehurst, D.H., Vogel, R., Paige, R.F. (eds.) Model Driven Architecture—Foundations and Applications (ECMDA-FA'07). Lecture Notes in Computer Science, vol. 4530, pp. 144–156. Springer, Berlin (2007)
139. Olsson, T., Grundy, J.: Supporting traceability and inconsistency management between software artefacts. In: Proceedings of the 2002 IASTED International Conference on Software Engineering and Applications. ACTA Press, Anaheim (2002). ISBN 978-0-88986-323-1
140. Ozkaya, I.: Representing requirement relationships. In: 1st International Workshop on Requirements Engineering Visualization (REV'06). IEEE Computer Society, New York (2006)

141. Paige, R.F., Olsen, G.K., Kolovos, D.S., Zschaler, S., Power, C.: Building model-driven engineering traceability classifications. In: ECMDA Traceability Workshop (ECMDA-TW) 2008 Proceedings, pp. 49–58. Sintef, Trondheim (2008). ISBN 978-82-14-04396-9
142. Palmer, J.D.: Traceabiliy. In: Thayer, R.H., Dorfman, M. (eds.) Software Requirements Engineering, 2nd edn, pp. 364–374. IEEE Computer Society Press, Los Alamitos (1997)
143. Pierce, R.A.: A requirements tracing tool. In: Proceedings of the Software Quality Assurance Workshop on Functional and Performance Issues, pp. 53–60. ACM, New York (1978)
144. Pinheiro, F.A.C.: Design of a Hyper-Environment for Tracing Object-Oriented Requirements. Ph.D. thesis, University of Oxford (1996)
145. Pinheiro, F.A.C.: Requirements traceability. In: Sampaio do Prado Leite, J.C., Doorn, J.H. (eds.) Perspectives on Software Requirements, pp. 93–113. Springer, Berlin (2003)
146. Pinheiro, F.A., Goguen, J.A.: An object-oriented tool for tracing requirements. IEEE Softw. **13**(2), 52–64 (1996)
147. Pohl, K.: PRO-ART: enabling requirements pre-traceability. In: 2nd International Conference on Requirements Engineering (ICRE'96) Proceedings, pp. 76–84. IEEE Computer Society, New York (1996)
148. Pohl, K.: Process-Centered Requirements Engineering. Wiley, New York (1996). ISBN 978-0-863-80193-8
149. Pohl, K., Haumer, P.: HYDRA: a hypertext model for structuring informal requirements representations. In: Requirements Engineering: Foundation for Software Quality (REFSQ'95) Proceedings. http://refsq.org (1995)
150. Potts, C., Bruns, G.: Recording the reasons for design decisions. In: ICSE '88: Proceedings of the 10th International Conference on Software Engineering, pp. 418–427. IEEE Computer Society, New York (1988)
151. Pressman, R.S.: Software Engineering: A Practitioner's Approach, 6th edn. McGraw-Hill, New York (2004). ISBN 978-0-073-01933-8
152. Ramamoorthy, C.V., Garg, V., Prakash, A.: Support for reusability in genesis. IEEE Trans. Softw. Eng. **14**(8), 1145–1154 (1988)
153. Ramesh, B.: Factors influencing requirements traceability practice. Commun. ACM **41**(12), 37–44 (1998)
154. Ramesh, B., Dhar, V.: Supporting systems development by capturing deliberations during requirements engineering. IEEE Trans. Softw. Eng. **18**(6), 498–510 (1992)
155. Ramesh, B., Edwards, M.: Issues in the development of a requirements traceability model. In: Proceedings of the IEEE International Symposium on Requirements Engineering, pp. 256–259. IEEE Computer Society, New York (1993)
156. Ramesh, B., Jarke, M.: Toward reference models for requirements traceability. IEEE Trans. Softw. Eng. **27**(1), 58–93 (2001)
157. Ramesh, B., Powers, T., Stubbs, C., Edwards, M.: Implementing requirements traceability: a case study. In: 2nd IEEE International Symposium on Requirements Engineering (RE'95) Proceedings, pp. 89–95. IEEE Computer Society, New York (1995)
158. Ramesh, B., Stubbs, C., Powers, T., Edwards, M.: Requirements traceability: theory and practice. Ann. Softw. Eng. **3**, 397–415 (1997)
159. Ramsin, R., Paige, R.F.: Process-centered review of object oriented software development methodologies. ACM Comput. Surv. **40**(1), 3:1–3:89 (2008)
160. Reiss, S.P.: Incremental maintenance of software artifacts. In: 21st IEEE International Conference on Software Maintenance (ICSM'05), pp. 113–122. IEEE Computer Society, New York (2005)
161. Richardson, J., Green, J.: Automating traceability for generated software artifacts. In: Proceedings of the 2nd International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE '03), pp. 24–33. ACM, New York (2004)
162. Rummler, A., Grammel, B., Pohl, C.: Improving traceability in model-driven development of business applications. In: ECMDA Traceability Workshop (ECMDA-TW) 2007 Proceedings, pp. 7–15. Sintef, Trondheim (2007). ISBN 978-82-14-04056-2
163. Sabetzadeh, M., Easterbrook, S.: Traceability in viewpoint merging: a model management perspective. In: TEFSE '05: Proceedings of the 3rd International Workshop on Traceability in Emerging Forms of Software Engineering, pp. 44–49. ACM, New York (2005)
164. Sherba, S.A.: Towards automating traceability: an incremental and scalable approach. Ph.D. thesis, University of Colorado at Boulder, USA (2005)
165. Sherba, S.A., Anderson, K.M., Faisal, M.: A framework for mapping traceability relationships. In: Proceedings of the 2nd International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE '03). ACM, New York (2003)
166. Simpson, J., Weiner, E. (eds.): Oxford English Dictionary, vol. 18, 2nd edn. Clarendon Press, Oxford (1989). ISBN 978-0-198-61186-8
167. Sinha, V., Sengupta, B., Chandra, S.: Enabling collaboration in distributed requirements management. IEEE Softw. **23**(5), 52–61 (2006)
168. Smith, M., Weiss, D., Wilcox, P., Dewar, R.: The OPHELIA traceability layer. In: 2nd Workshop on Cooperative Supports for Distributed Software Engineering Processes (CSSE 2003), pp. 150–161. FrancoAngeli, Benevento (2003)
169. Sousa, A., Kulesza, U., Rummler, A., Anquetil, N., Moreira, R.M.A., Amaral, V., Araújo, J.A.: A model-driven traceability framework to software product line development. In: ECMDA Traceability Workshop (ECMDA-TW) 2008 Proceedings, pp. 97–109. Sintef, Trondheim (2008). ISBN 978-82-14-04396-9
170. Spanoudakis, G., d'Avila Garcez, A.S., Zisman, A.: Revising rules to capture requirements traceability relations: a machine learning approach. In: 15th International Conference in Software Engineering and Knowledge Engineering (SEKE 2003) Proceedings, pp. 570–577. Knowledge Systems Institute, Skokie (2003). ISBN 978-1-891706-12-7
171. Spanoudakis, G., Zisman, A. Software traceability: a roadmap. In: Chang, S.K. (ed.) Handbook of Software Engineering and Knowledge Engineering, vol. 3—Recent Advances, pp. 395–428. World Scientific, Singapore (2005). ISBN 978-9-8125-6273-9
172. Spanoudakis, G., Zisman, A., Pérez-Miñana, E., Krause, P.: Rule-based generation of requirements traceability relations. J. Syst. Softw. **72**(2), 105–127 (2004)
173. Stirewalt, R.E.K., Deng, M., Cheng, B.H.C.: UML formalization is a traceability problem. In: TEFSE '05: Proceedings of the 3rd International Workshop on Traceability in Emerging Forms of Software Engineering, pp. 31–36. ACM, New York (2005)
174. Stone, A., Sawyer, P.: Finding tacit knowledge by solving the pre-requirements tracing problem. In: Requirements Engineering: Foundation for Software Quality (REFSQ'05) Proceedings. http://refsq.org (2005)
175. Stone, A., Sawyer, P.: Identifying tacit knowledge-based requirements. IEE Proc. Softw. **153**(6), 211–218 (2006)
176. Strašunskas, D.: Traceability in collaborative systems development from lifecycle perspective. In: Proceedings of the 1st International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE '02). ACM, New York (2002)
177. Strašunskas, D.: Domain model-centric distributed development: an approach to semantics-based change impact management. Ph.D. thesis, Norwegian University of Science and Technology (NTNU), Norway (2006)
178. Strašunskas, D., Hakkarainen, S.: Domain model driven approach to change impact assessment. In: Linger, H., Fisher, J.,

Wojtkowski, W.G., Wojtkowski, W., Zupančič, J., Vigo, K., Arnold, J. (eds.) Constructing the Infrastructure for the Knowledge Economy: Methods and Tools, Theory and Practice, pp. 305–316. Springer, Berlin (2004). ISBN 978-0-306-48554-1
179. Tilbury, A.J.M.: Enabling software traceability. In: IEE Colloquium on the Application of Computer Aided Software Engineering Tools, pp. 7/1–7/4. IEEE, London (1989)
180. Trainer, E., Quirk, S., de Souza, C., Redmiles, D.: Bridging the gap between technical and social dependencies with Ariadne. In: Eclipse Technology Exchange (eTX) Workshop. ACM, New York (2005)
181. van den Berg, K., Conejero, J.M., Hernández, J.: Analysis of crosscutting across software development phases based on traceability. In: EA '06: Proceedings of the 2006 International Workshop on Early Aspects at ICSE, pp. 43–50. IEEE Computer Society, New York (2006)
182. van Gorp, P., Altheide, F., Janssens, D.: Towards 2D traceability in a platform for contract aware visual transformations with tolerated inconsistencies. In: Enterprise Distributed Object Computing Conference, IEEE International, pp. 185–198. IEEE Computer Society, New York (2006)
183. van Gorp, P., Janssens, D.: CAViT: a consistency maintenance framework based on transformation contracts. In: Cordy, J.R., Lämmel, R., Winter, A. (eds.) Transformation Techniques in Software Engineering, no. 05161 in Dagstuhl Seminar Proceedings. LZI, Wadern (2006). ISSN 1862-4405
184. van Lamsweerde, A.: Formal specification: a roadmap. In: Proceedings of the Conference on The Future of Software Engineering (at ICSE '00), pp. 147–159. IEEE Computer Society, New York (2000)
185. van Lamsweerde, A.: Goal-oriented requirements engineering: a guided tour. In: 5th IEEE International Symposium on Requirements Engineering (RE '01) Proceedings. IEEE Computer Society, New York (2001)
186. von Pilgrim, J.: Mental map and model driven development. In: Fish, A., Knapp, A., Störrle, H. (eds.) Proceedings of the Workshop on the Layout of (Software) Engineering Diagrams (LED 2007), Electronic Communications of the EASST, vol. 7, pp. 17–32 (2007). ISSN 1863-2122
187. von Pilgrim, J., Vanhooff, B., Schulz-Gerlach, I., Berbers, Y.: Constructing and visualizing transformation chains. In: Model Driven Architecture—Foundations and Applications (ECMDA-FA'08). Lecture Notes in Computer Science, vol. 5095, pp. 17–32. Springer, Berlin (2008). ISBN 978-3-540-69095-5
188. van der Straeten, R., Mens, T., Simmonds, J., Jonckers, V.: Using description logic to maintain consistency between UML models. In: UML 2003: The Unified Modeling Language Proceedings. Lecture Notes in Computer Science, vol. 2863, pp. 326–340. Springer, Berlin (2003)
189. Vanhooff, B., Ayed, D., Van Baelen, S., Joosen, W., Berbers, Y.: UniTI: a unified transformation infrastructure. In: Engels, G., Opdyke, B., Schmidt, D.C., Weil, F. (eds.) Model Driven Engineering Languages and Systems, Proceedings of the 10th International Conference (MoDELS 2007). Lecture Notes in Computer Science, vol. 4735, pp. 31–45. Springer, Berlin (2007). ISBN 978-3-540-75208-0
190. Vanhooff, B., Baelen, S.V., Joosen, W., Berbers, Y.: Traceability as input for model transformations. In: ECMDA Traceability Workshop (ECMDA-TW) 2007 Proceedings, pp. 37–46. Sintef, Trondheim (2007). ISBN 978-82-14-04056-2
191. von Knethen, A.: Change-Oriented Requirements Engineering. Support for Evolution of Embedded Systems. Ph.D. thesis, Universität Kaiserslautern, Germany (2001)
192. von Knethen, A., Grund, M.: QuaTrace: a tool environment for (semi-) automatic impact analysis based on traces. In: Proceedings of the IEEE International Conference on Software Maintenance (ICSM 2003), pp. 246–255. IEEE Computer Society, New York (2003)
193. von Knethen, A., Paech, B.: A survey on tracing approaches in practice and research. Research Report 095.01/E, Fraunhofer IESE, Kaiserslautern, Germany (2002)
194. Walderhaug, S., Johansen, U., Stav, E., Aagedal, J.: Towards a generic solution for traceability in MDD. In: ECMDA Traceability Workshop (ECMDA-TW) 2006 Proceedings, pp. 41–50. Sintef, Trondheim (2006). ISBN 978-82-14-04030-2
195. Walderhaug, S., Stav, E., Johansen, U., Olsen, G.K.: Traceability in model-driven software development. In: Tiako, P.F. (ed.) Designing Software-Intensive Systems: Methods and Principles, pp. 133–159. Idea Group Publishing, Hershey (2008). ISBN: 978-1599046990
196. Watkins, R., Neal, M.: Why and how of requirements tracing. IEEE Softw. **11**(4), 104–106 (1994)
197. Wenzel, S., Hutter, H., Kelter, U.: Tracing model elements. In: 23rd International Conference on Software Maintenance (ICSM'07), pp. 104–113. IEEE Computer Society, New York (2007)
198. Whittle, J., Baalen, J.V., Schumann, J., Robinson, P., Pressburger, T., Penix, J., Oh, P., Lowry, M., Brat, G.: Amphion/NAV: deductive synthesis of state estimation software. In: ASE '01: Proceedings of the 16th IEEE International Conference on Automated Software Engineering, pp. 395–399. IEEE Computer Society, New York (2001)
199. Wiegers, K.E.: Software Requirements, 2nd end. Microsoft Press, Redmond (2003). ISBN 978-0-735-61879-4
200. Wieringa, R.: An introduction to requirements traceability. Tech. rep., Institute for Mathematics and Computer Science, Vrije Universiteit, Amsterdam, The Netherlands (1995)
201. Winkler, S.: On usability in requirements trace. Visualizations. In: 3rd International Workshop on Requirements Engineering Visualization (REV'08). IEEE Computer Society, New York (2008)
202. Zou, X., Settimi, R., Cleland-Huang, J.: Phrasing in dynamic requirements trace retrieval. In: 30th Annual International Computer Software and Applications Conference (COMPSAC'06), pp. 265–272. IEEE Computer Society, New York (2006)

Author Biographies

Stefan Winkler is a Ph.D. candidate at the department of mathematics and computer science of the FernUniversität in Hagen, Germany. His Ph.D. topic is about requirements traceability in the context of evolving artifacts. He received his computer science degree in the year 2003 from the University of Stuttgart, Germany. Beyond traceability, his main interests lie in the area of requirements engineering and modeling.

Jens von Pilgrim is a Ph.D. candidate at the department of mathematics and computer science of the FernUniversität in Hagen, Germany. His Ph.D. topic is about semi-automated transformations in the context of model-driven development. He received his computer science degree in 2005, after working as programmer and software architect in the industrie for more than 10 years.