A Quality Model for the Systematic Assessment of Requirements Traceability

Patrick Rempel
Technische Universität Ilmenau
Software Systems/Process Informatics Group
Ilmenau, Germany
patrick.rempel@tu-ilmenau.de

Patrick Mäder
Technische Universität Ilmenau
Software Systems/Process Informatics Group
Ilmenau, Germany
patrick.maeder@tu-ilmenau.de

***Abstract*—Traceability is an important quality of software requirements and allows to describe and follow their life throughout a development project. The importance of traceable requirements is reflected by the fact that requirements standards, safety regulations, and maturity models explicitly demand for it. In practice, traceability is created and maintained by humans, which make mistakes. In result, existing traces are potentially of dubious quality but serve as the foundation for high impact development decisions. We found in previous studies that practitioners miss clear guidance on how to systematically assess the quality of existing traces. In this paper, we review the elements involved in establishing traceability in a development project and derive a quality model that specifies per element the acceptable state (Traceability Gate) and unacceptable deviations (Traceability Problem) from this state. We describe and formally define how both, the acceptable states and the unacceptable deviations can be detected in order to enable practitioners to systematically assess their project's traceability. We evaluated the proposed model through an expert survey. The participating experts considered the quality model to be complete and attested that its quality criteria are of high relevance. We further found that the experts weight the occurrence of different traceability problems with different criticality. This information can be used to quantify the impact of traceability problems and to prioritize the assessment of traceability elements.**

***Index Terms*—requirements traceability; traceability problem classes; problem dependencies; assessment; inspection**

I. Introduction

Requirements traceability, defined as the "ability to describe and follow the life of a requirement in both forwards and backwards direction" [1], is a critical element of any rigorous software development process. It provides support for numerous software engineering tasks such as requirements validation, impact analysis, coverage analysis, compliance verification, and derivation analysis. The importance of requirements traceability is also reflected by the fact that it is explicitly demanded by a widely accepted requirements quality model [2]. Furthermore, the IEEE standard 830-1998 [3] on recommended practices for software requirements specification as well as the ISO-29148:2011 standard [4] on requirements engineering state that requirements should be traceable.

However, Cleland-Huang et al. [5] state that achieving purposed and trustworthy traceability remains a challenge, which is not yet solved. In industrial practice, traceability is created and maintained by humans who make mistakes [6]. The resulting traceability is often of dubious quality but serves as the foundation for high impact development decisions. Developing safety-critical systems, for example, requires the compilation of a safety case to argue that as system is safe for use [7]. A safety argument is typically made through traceability [8]. Thereby, traceability is used to provide evidence that all safety requirements are implemented and verified, and its originating hazard or threat is documented. In a number of previous studies [9], [10], [6], [11], [12] we investigated the quality of industrially captured and used traceability data and found that all software projects assessed across all our studies struggled with problems such as a lack of compliance or latent safety risk, which were caused by incomplete or missing traceability data. When reflecting on the results of those previous studies, we realized that no classification of traceability problems was available to systematically assess traceability for structural deficiencies. This lack of a well defined problem classification makes it difficult for software practitioners to generate an understanding of possible traceability problems and how to recognize them.

To approach this problem, we followed a two stage process. First, we systematically review the elements that are required to plan and establish traceability in a software development project. Second, we propose a quality model that specifies relevant quality attributes. Driven by these quality attributes, we define for each element an acceptable state (traceability gate) and unacceptable deviations (traceability problem) from this state through generalized formal expressions. We also provide criteria for how traceability elements in software development projects can be assessed systematically in order to detect existing structural traceability problems. To evaluate the completeness and practical relevance of our proposed model, we conducted a traceability experts survey.

The remainder of the paper is organized as follows. In Section II, we introduce fundamental elements of requirements traceability and discuss the assumptions of our work. In Section III, we present a quality model to assess the functional suitability of traceability in software development projects. To evaluate this proposed assessment model, we conducted a traceability experts survey described in Section IV. Section V reviews related work in the area of requirements traceability and, more specifically, in systematically assessing software project's traceability data. In Section VI, we discuss the main results, contributions, and limitations of our work. Based on the limitations, we outline future research directions.

II. Preliminary Assumptions and Definitions

The traceability life cycle of a software development project comprises the following processes: planning and managing a traceability strategy, creating and maintaining traceability data, and using traceability data [13]. These processes deal with a variety of traceability elements that may suffer from structural problems and can prevent a proper usage. In this section, we provide an overview of these elements.

A. Elements of the Traceability Planning Stage

Traceability planning refers to the specification of traceable artifact classes and of permissible trace link classes between pairs of those artifact classes [14], [13]. As traceability relations can be established through a direct trace link or a sequence of trace links, an additional trace path class concept is necessary [12]. The output of the planning is a so-called Traceability Information Model (TIM) [15], which is an accepted means for communicating planning results [16], [17], [18], [19]. The upper part of Figure 1 shows the meta-model of this TIM. The lower part of Figure 1 exemplifies TIM elements and introduces a graphical notation for those elements, used throughout this paper.

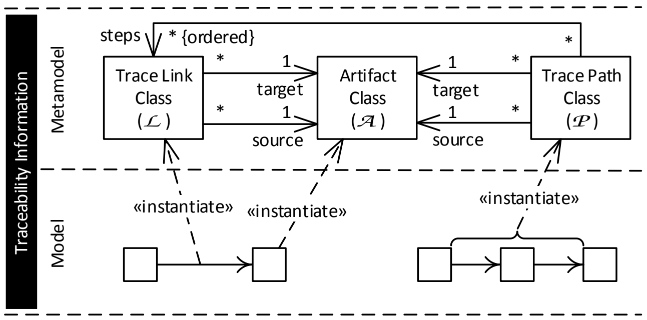

Traceability Information
Metamodel
Model
steps
* {ordered}
Trace Link Class ($\mathcal{L}$)
Artifact Class ($\mathcal{A}$)
Trace Path Class ($\mathcal{P}$)
target
source
«instantiate»

Fig. 1. Overview of the Traceability Information Model (TIM) elements

An *artifact class* specifies classes of artifacts for which traceability is required within the software development project, denoted as $\mathcal{A}$. For example, a TIM planning traceability between specification and testing would at least consist of the two artifact classes `requirement` and `test case`.

A *trace link class* specifies classes of trace links between two artifact classes. Let $\mathcal{L}$ be the set of all defined trace link classes. In this paper, we denote a trace link class as $\mathcal{A} \xrightarrow{\mathcal{L}} \mathcal{A}$. The originating artifact class holds the role source and the destination artifact class holds the role target. The function $f_s : \mathcal{L} \rightarrow \mathcal{A}$ maps each trace link class to its source artifact class. The function $f_t : \mathcal{L} \rightarrow \mathcal{A}$ maps each trace link class to its target artifact class. For example, a trace link class `requirement` $\xrightarrow{verify}$ `test case` implies that $f_s(verify) =$ `requirement` and $f_t(verify) =$ `test case`.

A *trace path class* specifies a sequence of trace link classes between two artifact classes. Let $\mathcal{P}$ be the set of all defined trace path classes. We denote a trace path class as $\mathcal{A} \xrightarrow{\mathcal{L}} \mathcal{A}$ ... $\mathcal{A} \xrightarrow{\mathcal{L}} \mathcal{A}$. As depicted in the meta-model in Figure 1, a trace path class defines a directed transitive association category between the two related artifact classes through an ordered sequence of trace link classes. Each trace link class of the sequence refers to one segment within the trace path class. Similar to a trace link class, the originating artifact class holds the role source and the destination artifact class holds the role target. The function $f_s : \mathcal{P} \rightarrow \mathcal{A}$ maps each trace path class to its source artifact class. The function $f_t : \mathcal{P} \rightarrow \mathcal{A}$ maps each trace path class to its target artifact class. For example, a software project intending to establish traceability between the two artifact classes `requirement` and `test result` to ensure that every requirement is implemented correctly would define a trace path class `requirement` $\xrightarrow{verify}$ `test case` $\xrightarrow{result}$ `test result`, because `requirements` can only be traced to `test results` through the intermediate artifact class `test case`. This implies that $f_s(\langle verify, result \rangle) =$ `requirements` and $f_t(\langle verify, result \rangle) =$ `test results`.

B. Elements of the Traceability Instantiation Stage

In order to use traceability, it is required to establish trace links as specified in the TIM usually by means of an explicit registration of the artifacts and their links in a traceability repository [14]. Gotel et al. [13] define the traceability creation process as "*an activity of associating two (or more) artifacts, by providing trace links between them, for tracing purposes*". Traceability maintenance is defined by the same authors [13] as "*activity associated with updating pre-existing trace links as changes are made to the traced artifacts and the traceability evolves, creating new trace links where needed to keep the traceability relevant and up to date*". In this paper, we use the concept Traceability Data Model (TDM) to refer to all traceability data that is created, maintained, and used throughout a development lifecycle. To clearly define the fundamental elements of traceability creation and maintenance, we define a meta-model for the TDM, which consists of the elements *artifact*, *trace link*, and *trace path*. The upper part of Figure 2 depicts this meta-model, while the lower part introduces our graphical notation for instances of TDM's elements.

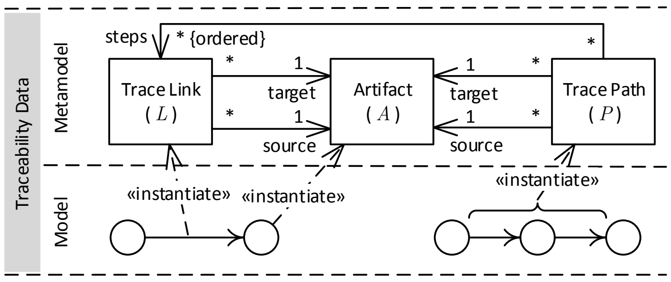


Fig. 2. Overview of the Traceability Data Model (TDM) elements

An *artifact* refers to any output, whether final or not, that is produced or maintained throughout the software life-cycle. Let A be the set of all artifacts. It exists an instantiation relationship between the artifact class and the artifact (see

Figure 3), where the artifact class element holds the role classifier and the artifact element holds the role object. The function $f_i : \mathcal{A} \rightarrow 2^A$ maps each artifact class a_i to a set of artifacts, which are instances of a_i so that $f_i(a_i) = \{\text{instances of the artifact class } a_i\}$. For example, a project developing an Autopilot system specifies a `requirement` r_1 stating that "*altitude climb control is entered whenever* $|$*actual altitude* $-$ *desired altitude*$| \leq 1500$", a `test case` tc_1 supposed to verify r_1, and a `test result` tr_1 documenting the result of the executed test. This means that r_1, tc_1, and tr_1 are project artifacts ($r_1, tc_1, tr_1 \in A$). Thereby, r_1 is an instance of the artifact class `requirement` $f_i(\texttt{requirement}) = \{r_1\}$, tc_1 is an instance of the artifact class `test case` $f_i(\texttt{test case}) = \{tc_1\}$, and tr_1 is an instance of the artifact class `test result` $f_i(\texttt{test result}) = \{tr_1\}$.

A *trace link* is a directed association established between two artifacts. Let L be the set of all trace links. We denote a trace link as $A \xrightarrow{L} A$. The originating artifact holds the role source and the destination artifact holds the role target. The function $f_s : L \rightarrow A$ maps each trace link to its source artifact. The function $f_t : L \rightarrow A$ maps each trace link to its target artifact. It exists an instantiation relationship between the trace link class and the trace link (see Figure 3), where the trace link class element holds the role classifier and the trace link element holds the role object. The function $f_i : \mathcal{L} \rightarrow 2^L$ maps each trace link class l_i to a set trace links, which are instances of l_i so that $f_i(l_i) = \{\text{instances of the trace link class } l_i\}$. For example, a developed Autopilot system could contain a trace link $r_1 \xrightarrow{l_1[verify]} tc_1$ to document that r_1 is verified by tc_1.

A *trace path* refers to a sequence of trace links that establish a transitive relationship between two artifacts. Let P be the set of all trace paths. We denote a trace path as $A \xrightarrow{L} A \ldots A \xrightarrow{L} A$. Each trace link of the sequence holds the role of one segment within the trace path. Similar to a trace link, the originating artifact holds the role source and the destination artifact holds the role target. The function $f_s : P \rightarrow A$ maps each trace path to its source artifact. The function $f_t : P \rightarrow A$ maps each trace path to its target artifact. It exists an instantiation relationship between the trace path class and the trace path (see Figure 3), where the trace path class element holds the role classifier and the trace path element holds the role object. The function $f_i : \mathcal{P} \rightarrow 2^P$ maps each trace path class p_i to a set trace paths, which are instances of p_i so that $f_i(p_i) = \{\text{instances of the trace path class } p_i\}$. For example, a developed Autopilot system could contain a trace path $r_1 \xrightarrow{l_1[verify]} tc_1 \xrightarrow{l_2[result]} tr_1$ to document that tr_1 is the verification result of r_1.

C. Traceability Roles and their Stakes in a Project

Various actors are involved in the process of planning traceability information as well as in the process of establishing traceability data. In this section, we characterize the roles of these actors. We refer to the role that is concerned with the planning of traceability information as *traceability planner*. A traceability planner is responsible for specifying a

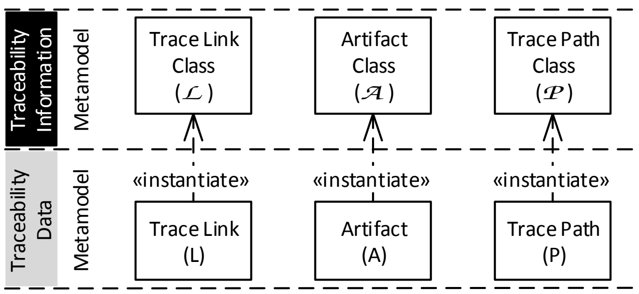


Fig. 3. Overview of the dependencies between elements of the Traceability Information Model (TIM) and the Traceability Data Model (TDM)

project-specific TIM, which addresses the project's traceability concerns. Gotel et al. [13] argue that the traceability planner perspective "*demands understanding [..] stakeholders who may need [..] traceability*". Accordingly, we refer to the role that is demanding traceability between artifact classes as *traceability stakeholder*. The role of a traceability stakeholder can be fulfilled by a variety of actors such as customers, regulation authorities, project managers, and developers. We refer to the role that is concerned with establishing traceability data as the *traceability creator*. A traceability creator is responsible for associating artifacts via trace links to implement a project's traceability data. These traceability data are used by the *traceability stakeholder* in order to support software and systems engineering tasks [13]. In Figure 4, we summarize the three traceability roles and their main activities.

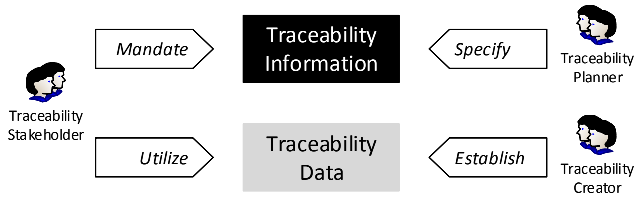


Fig. 4. Overview of the three traceability roles and their stakes in a project's traceability information and traceability data

Traceability roles have different stakes in a project's traceability information and traceability data. Based on these stakes, we distinguish different categories of traceability elements. We denote traceability information, which is mandated by the role traceability stakeholder as mandated elements of traceability planning. Let $O_{\mathcal{A}}$ bet the set of all mandated artifact classes, $O_{\mathcal{L}}$ be the set of all mandated trace link classes, and $O_{\mathcal{P}}$ be the set of all mandated trace path classes, which are mandated by the role traceability stakeholder. Furthermore, we denote traceability information that is specified by the role traceability planner as specified elements of traceability planning. Let $S_{\mathcal{A}}$ bet the set of all Specified Artifact Classes, $S_{\mathcal{L}}$ be the set of all specified trace link classes, and $S_{\mathcal{P}}$ be the set of all specified trace path classes, which are specified by the role traceability planner. Finally, we denote traceability data that is established by the role traceability creator as existing elements of traceability. Let $\mathcal{E}_A$ bet the set of all Existing artifacts,

$\mathcal{E}_L$ be the set of all existing trace links, and $\mathcal{E}_P$ be the set of all existing trace paths, which are established by the role traceability creator. The different types of traceability elements are summarized in a Venn diagram in Figure 5. The traceability role symbols in Figure 5 illustrate, which set of traceability elements is produced by which traceability role.

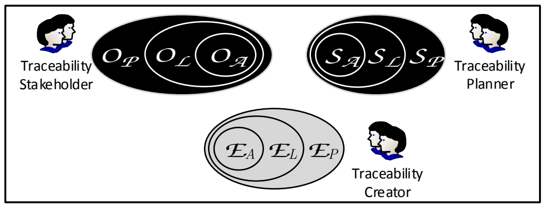


Fig. 5. Overview of the types of traceability elements by traceability roles as Venn diagram

III. Quality Model for Traceability Assessment

A variety of similar definitions for the term quality exists, provided by institutions like ISO, IEC, and IEEE. The ISO 9000:2005 standard [20] defines quality as "*degree to which a set of inherent characteristics fulfills requirements*". By adapting this definition, we define traceability quality as the degree to which existing artifacts of a software development project are traceable as mandated by the project's traceability stakeholders. In recent empirical studies on the quality of traceability in software development projects [10], [6], [12], we found that all of them suffered from the problem that the established traceability was not suitable for its intended purpose. Practitioners participating in our empirical studies reported that they miss clear guidance on how to systematically assess their implemented traceability. To address this problem, we propose a Traceability Assessment Model (TAM) for analyzing the suitability of established traceability with respect to its intended purpose. Figure 6 shows the meta-model of the proposed TAM. The following paragraphs discuss each model element with respect to its purpose.

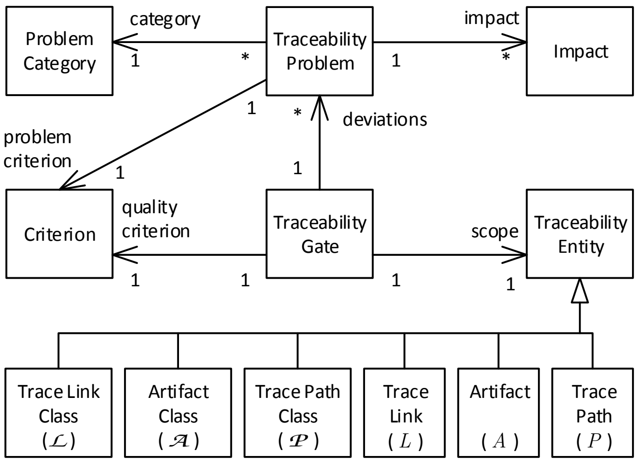


Fig. 6. Metamodel of the Traceability Assessment Model (TAM)

A *traceability entity* represents the object in question for which the quality is to be assessed through the proposed model. In particular, the fundamental traceability elements are artifact classes, trace link classes, trace path classes, artifacts, trace links, and trace paths (see Section II). Let E be the set of all traceability entities, where $E \subseteq \mathcal{A} \cup \mathcal{L} \cup \mathcal{P} \cup A \cup L \cup P$. The set of all traceability entities represents the subject of the quality assessment. Hence, our proposed model clearly specifies its focus, which is the set of all traceability entities. The context is the software development project, where the traceability entities are created and used.

A *traceability gate* element specifies the acceptable state of a traceability entity in terms of quality. To provide clear and unambiguous assessment guidance for practitioners, each traceability gate has assigned one quality criterion, which is represented by the criterion element. The assigned quality criterion defines, how the acceptable state of a traceability entity can be determined by an assessment and is specified as a formal expression. Let Q be the set of all defined traceability gates and C be the set of all defined criteria.

A traceability entities may deviate from its acceptable state. This unacceptable deviation is represented in the TAM as a traceability problem element. Let $\bar{Q}$ be the set of all traceability problems. Analogous to the traceability gate, each traceability problem has assigned one problem criterion, which is represented by the criterion element and provides guidance for practitioners in detecting problems. Since different kinds of traceability problems can exist per traceability gate, we assign a problem category to each traceability problem. This problem categorization is supposed to improve the model comprehensiveness and can also support practitioners with clustering traceability problems, for example in situations were some problem categories are relevant for a specific traceability assessment only. To support the interpretation of the assessment results, each traceability problem has assigned one impact element, which provides explanation to practitioners on what impact to be expected for the development project from the detected deviation.

A. The Traceability Assessment Model (TAM) Explained

Traceability quality is defined as the degree to which existing artifacts in a software development project are traceable as mandated by the project's traceability stakeholders. Hence, traceability assessment is about determining this degree of traceability fulfillment.

Since the mandated traceability is implemented by two roles throughout the traceability life cycle, the traceability planner and the traceability creator (see also Section II-C), two dimensions of potential quality problems can be derived. First, the traceability information, which is specified by the traceability planner, does not conform with the mandated traceability. Second, the traceability data, which is established by the traceability creator, does not conform with the specified traceability information.

Each assessment can be characterized by the fact that two sets of data are compared for conformance. One set contains required data and the other set contains the fulfilling data. Hence, a conformance assessment can lead to four possible results. First, the fulfilling set *fully conforms* with the set of required data. This kind of result is represented in our quality model by the traceability gate elements. Second, the fulfilling set is *incomplete*, because it misses data in order to fully conform with the set of *required* data. This state is represented in our quality model by the problem category *missing traceability*. Third, the fulfilling set is *redundant*, because it contains superfluous data, not necessary to conform with the set of *required* data. This state is represented in our quality model by the problem category *superfluous traceability*. Fourth, the fulfilling set is *incomplete* and *redundant*, and thus a composite problem categories of *missing traceability* and *superfluous traceability*.

By combining the complete enumeration of potential traceability quality problems with the complete enumeration of possible conformance assessment results, we derived a TAM, which is summarized in Table I. While the first three rows cover the conformance assessment of mandated versus specified traceability information, the last three rows cover the conformance assessment of existing traceability data versus specified traceability information. The first column contains traceability gate definitions for each traceability entity, the second column contains traceability problem definitions of the problem category missing traceability, and the third column contains traceability problem definitions of the problem category superfluous traceability for each traceability entity. For better readability, the criteria of each traceability gate and traceability problem are defined as formal expression and textual description. Additionally, each criteria is graphically illustrated as Venn diagram. The table shows that the TAM does not contain a case for superfluous artifacts. It was omitted deliberately, because an artifact is an instance of an artifact classifier and by definition cannot exist without it.

B. Traceability Problem Impacts

We finalize the discussion of the proposed TAM by providing a qualitative discussion of the potential impacts of traceability problems. A quantitative discussion of the potential impacts is provided in Section IV, where we present the results of a traceability experts survey with 13 subjects.

In Figure 7, a graph of all interdependencies between traceability problems is shown. This graph illustrates that the interdependencies between traceability problems of the category superfluous are similar to those between traceability problems of the category missing, because they result from the same dependencies between traceability entities.

Missing traceability information: The traceability problems $\boxed{1_M}$ missing artifact class, $\boxed{2_M}$ missing trace link class, and $\boxed{3_M}$ missing trace path class imply that the specified traceability information within an assessed software development project is incomplete. Since the traceability entities defined in the TDM's and TIM's meta-models depend upon another (see Figures 1, 2, and 3), also the derived problem classes depend upon another. In particular, the problem $\boxed{1_M}$ implies $\boxed{2_M}$, because an artifact class is an integral component of a trace link class. However, $\boxed{1_M}$ only indicates potentially $\boxed{3_M}$, because an artifact class is an integral component of a trace path class as well, but for one specific artifact class a trace path class might not be required. Since each artifact is an instance of artifact class, $\boxed{1_M}$ also implies $\boxed{4_M}$. Similarly, $\boxed{2_M}$ implies $\boxed{5_M}$ and $\boxed{3_M}$ implies $\boxed{6_M}$, because of the respective instantiate dependencies. The problem $\boxed{3_M}$ implies $\boxed{2_M}$, because an trace link class is an integral component of a trace path class.

Missing traceability data: The traceability problems $\boxed{4_M}$ missing artifact, $\boxed{2_M}$ missing trace link, and $\boxed{6_M}$ missing trace path imply that the existing traceability data within the assessed software development project is incomplete. Based on existing traceability entity dependencies, the problem $\boxed{4_M}$ implies $\boxed{5_M}$, because an artifact is an integral component of a trace link. However, $\boxed{4_M}$ only indicates potentially $\boxed{6_M}$, because an artifact is an integral component of a trace path as well, but for one specific artifact a trace path might not be required. $\boxed{6_M}$ implies $\boxed{5_M}$, because a trace link is an integral component of a trace path.

Superfluous traceability information: The traceability problems $\boxed{1_S}$ superfluous Artifact Class, $\boxed{2_S}$ superfluous trace link class, and $\boxed{3_S}$ superfluous trace path class imply that there exist elements in the specified traceability information of the assessed software development project, which are not necessary for any traceability related purpose. Eliminating those unnecessary elements reduces the effort spend on establishing traceability because the creation and maintenance the superfluous trace links and trace paths would be omitted. Again, the problem classes depend upon another. Problem $\boxed{1_S}$ indicates a $\boxed{2_S}$ problem and a $\boxed{3_S}$ problem, because an artifact class is an integral part of trace link classes and trace path classes. It is only an indication, because theoretically an artifact class could be specified without participating in any trace link class. Since each trace link is an instance of a trace link class, $\boxed{2_S}$ implies $\boxed{5_S}$. Similarly, $\boxed{3_S}$ implies $\boxed{6_S}$. The problem $\boxed{3_S}$ implies $\boxed{2_S}$, because an trace link class is an integral component of a trace path class.

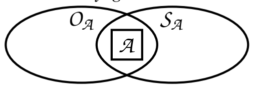
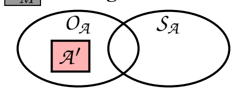
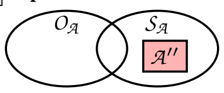
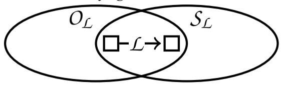
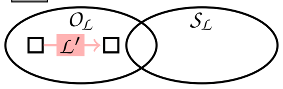
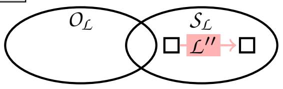
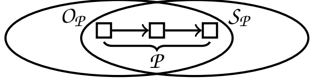
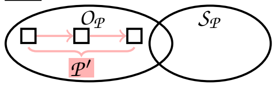
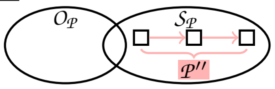
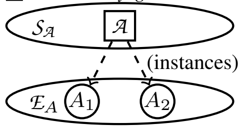
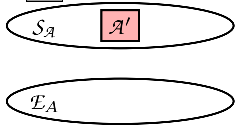
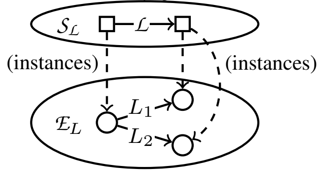
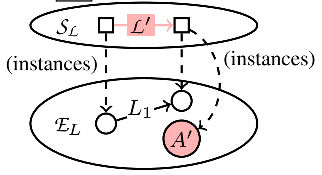
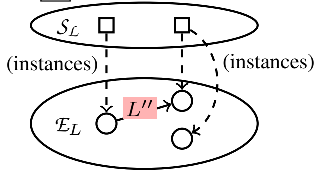
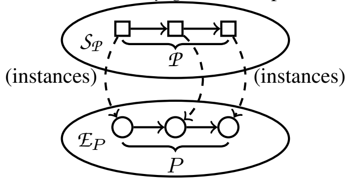
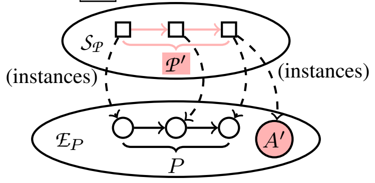
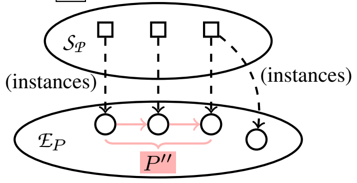
Superfluous traceability data: The traceability problems $\boxed{2_S}$ superfluous trace link and $\boxed{6_S}$ superfluous trace path imply that the existing traceability data in the assessed software development project are not necessary. Accordingly, the creation can be omitted to reduce traceability implementation effort.

IV. Traceability Experts Survey

In this section, we discuss an empirical validation of the proposed TAM. Our main goal for proposing this model is to provide guidance for practitioners in systematically assessing the quality of existing traceability in a software development project. Hence, we focus our validation on the model's structural dimension, which is related to its intended use. We conducted a survey with 13 traceability experts to investigate the completeness and usefulness of the proposed model. The participating subjects had an average practical software traceability experience of 8.6 years. Additionally, 12 out of the 13 participants had either worked in a project that

TABLE I

COMPLETE OVERVIEW OF THE PROPOSED TRACEABILITY ASSESSMENT MODEL (TAM), SHOWING AN ENUMERATION OF THE TRACEABILITY GATES AND THE TRACEABILITY PROBLEMS FOR ALL TRACEABILITY ENTITIES

		traceability gates $\boxed{1}$ – $\boxed{6}$	**traceability problems** $\boxed{1_M}$ – $\boxed{6_S}$ Missing traceability	 Superfluous traceability
Traceability Information Model (TIM)	artifact class ($\mathcal{A}$)	$\boxed{1}$ *traceability gate*: artifact class  $\forall \mathcal{A}[\mathcal{A} \in (O_{\mathcal{A}} \wedge S_{\mathcal{A}})]$ Every mandated artifact class should be specified in the TIM and every artifact class specified in the TIM should be mandated.	$\boxed{1_M}$ **missing artifact class**  $\exists \mathcal{A}'[\mathcal{A}' \in (O_{\mathcal{A}} \setminus S_{\mathcal{A}})]$ There exists a mandated artifact class $\mathcal{A}'$, which is not specified within the TIM.	$\boxed{1_S}$ **superfluous Artifact Class**  $\exists \mathcal{A}''[\mathcal{A}'' \in (S_{\mathcal{A}} \setminus O_{\mathcal{A}})]$ The TIM specifies an artifact class $\mathcal{A}''$, which is not mandated.
	trace link class ($\mathcal{L}$)	$\boxed{2}$ *traceability gate*: trace link class  $\forall \mathcal{L}[\mathcal{L} \in (O_{\mathcal{L}} \wedge S_{\mathcal{L}})]$ Every mandated trace link class should be specified in the TIM and every trace link class specified in the TIM should be mandated.	$\boxed{2_M}$ **missing trace link class**  $\exists \mathcal{L}'[\mathcal{L}' \in (O_{\mathcal{L}} \setminus S_{\mathcal{L}})]$ There exists a mandated trace link class $\mathcal{L}'$, which is not specified within the TIM.	$\boxed{2_S}$ **superfluous trace link class**  $\exists \mathcal{L}''[\mathcal{L}'' \in (S_{\mathcal{L}} \setminus O_{\mathcal{L}})]$ The TIM specifies an trace link class $\mathcal{L}''$, which is not mandated.
	trace path class ($\mathcal{P}$)	$\boxed{3}$ *traceability gate*: trace path class  $\forall \mathcal{P}[\mathcal{P} \in (O_{\mathcal{P}} \wedge S_{\mathcal{P}})]$ Every mandated trace path class should be specified in the TIM and every trace path class specified in the TIM should be mandated.	$\boxed{3_M}$ **missing trace path class**  $\exists \mathcal{P}'[\mathcal{P}' \in (O_{\mathcal{P}} \setminus S_{\mathcal{P}})]$ There exists a mandated trace path class $\mathcal{P}'$, which is not specified within the TIM.	$\boxed{3_S}$ **superfluous trace path class**  $\exists \mathcal{P}''[\mathcal{P}'' \in (S_{\mathcal{P}} \setminus O_{\mathcal{P}})]$ The TIM specifies an trace path class $\mathcal{P}''$, which is not mandated.
Traceability Data Model (TDM)	artifact (A)	$\boxed{4}$ *traceability gate*: artifact  $S_{\mathcal{A}}$ $\mathcal{A}$ (instances) $\mathcal{E}_A$ A_1 A_2 $\forall \mathcal{A}[f_i(\mathcal{A}) \neq \varnothing]$ Every artifact class specified in the TIM should be instantiated in the TDM as one or many artifacts.	$\boxed{4_M}$ **missing artifact**  $\exists \mathcal{A}'[f_i(\mathcal{A}') = \varnothing]$ There exists an artifact class $\mathcal{A}'$ specified in the TIM for which no artifact instances exits in the TDM.	▨
	trace link (L)	$\boxed{5}$ *traceability gate*: trace link  $S_{\mathcal{L}}$ $\mathcal{L}$ (instances) (instances) $\mathcal{E}_L$ L_1 L_2 $\forall \mathcal{L} \forall A \subseteq f_i(f_s(\mathcal{L}) \cup f_t(\mathcal{L})) \exists L[A \in (f_s(L) \cup f_t(L)) \wedge f_s(L \subseteq f_i(f_s(\mathcal{L}) \wedge f_t(L \subseteq f_i(f_t(\mathcal{L})]$ Every artifact in the TDM, which is an instance of a source or target of a trace link class $\mathcal{L}$, should have one or many trace links L that instantiate the $\mathcal{L}$.	$\boxed{5_M}$ **missing trace link**  $S_{\mathcal{L}}$ $\mathcal{L}'$ (instances) (instances) $\mathcal{E}_L$ L_1 A' $\forall \mathcal{L}' \exists A' \subseteq f_i(f_s(\mathcal{L}) \cup f_t(\mathcal{L})) \neg \exists L'[A' \in (f_s(L') \cup f_t(L')) \wedge f_s(L' \subseteq f_i(f_s(\mathcal{L}') \wedge f_t(L' \subseteq f_i(f_t(\mathcal{L}')]$ There exists an artifact A' in the TDM which is an instance of a source or target of a trace link class $\mathcal{L}'$, but has no trace link that satisfies $\mathcal{L}'$.	$\boxed{5_S}$ **superfluous trace link**  $S_{\mathcal{L}}$ (instances) (instances) $\mathcal{E}_L$ L'' $\exists L'' \neg \exists \mathcal{L}''[f_s(L'' \subseteq f_i(f_s(\mathcal{L}'') \wedge f_t(L'' \subseteq f_i(f_t(\mathcal{L}'')]$ There exists a trace link L'' in the TDM, which does not satisfy any trace link class of the TIM.
	trace path (P)	$\boxed{6}$ *traceability gate*: trace path  $S_{\mathcal{P}}$ $\mathcal{P}$ (instances) (instances) $\mathcal{E}_P$ P $\forall \mathcal{P} \forall A \subseteq f_i(f_s(\mathcal{P}) \cup f_t(\mathcal{P})) \exists P[A \in (f_s(P) \cup f_t(P)) \wedge f_s(P \subseteq f_i(f_s(\mathcal{P}) \wedge f_t(P \subseteq f_i(f_t(\mathcal{P})]$ Every artifact in the TDM, which is an instance of a source or target of a trace path class $\mathcal{P}$, should have one or many trace paths P that instantiate the $\mathcal{P}$.	$\boxed{6_M}$ **missing trace path**  $S_{\mathcal{P}}$ $\mathcal{P}'$ (instances) (instances) $\mathcal{E}_P$ P A' $\forall \mathcal{P}' \exists A' \subseteq f_i(f_s(\mathcal{P}) \cup f_t(\mathcal{P})) \neg \exists P'[A' \in (f_s(P') \cup f_t(P')) \wedge f_s(P' \subseteq f_i(f_s(\mathcal{P}') \wedge f_t(P' \subseteq f_i(f_t(\mathcal{P}')]$ There exists an artifact A' in the TDM which is an instance of a source or target of a trace path class $\mathcal{P}'$, but has no trace path that satisfies $\mathcal{P}'$.	$\boxed{6_S}$ **superfluous trace path**  $S_{\mathcal{P}}$ (instances) (instances) $\mathcal{E}_P$ P'' $\exists P'' \neg \exists \mathcal{P}''[f_s(P'' \subseteq f_i(f_s(\mathcal{P}'') \wedge f_t(P'' \subseteq f_i(f_t(\mathcal{P}'')]$ There exists a trace path P'' in the TDM, which does not satisfy any trace path class of the TIM.

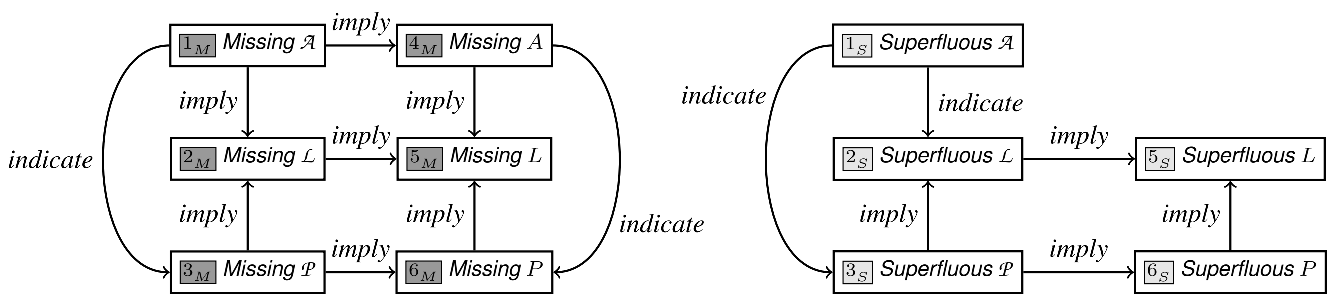

$\boxed{1_M}$ Missing $\mathcal{A}$
imply
$\boxed{4_M}$ Missing A
imply
imply
indicate
$\boxed{2_M}$ Missing $\mathcal{L}$
imply
$\boxed{5_M}$ Missing L
imply
imply
indicate
$\boxed{3_M}$ Missing $\mathcal{P}$
imply
$\boxed{6_M}$ Missing P
$\boxed{1_S}$ Superfluous $\mathcal{A}$
indicate
indicate
$\boxed{2_S}$ Superfluous $\mathcal{L}$
imply
$\boxed{5_S}$ Superfluous L
imply
imply
$\boxed{3_S}$ Superfluous $\mathcal{P}$
imply
$\boxed{6_S}$ Superfluous P

Fig. 7. Overview of the impacts among the traceability problems

captured traceability information or had previously assessed the quality of captured traceability information in a software project. Ten participants answered that they had done both.

Initially, we assembled a list of 34 potential participants, which were known to the authors from previous industrial traceability case studies and research collaborations. The authors discussed the suitability of each potential participant with regards to the purpose of the intended study. If one author doubted on a potential participant, she or he was discarded. Eventually, we obtained a list of 22 potential participants. We requested those 22 participants via e-mail to fill-in our questionnaire. Over a period of two weeks, we received 13 responses. The result data of the survey and the questionnaire is publicly available[1].

A. Data Collection

The questionnaire comprised five parts: 1) introduction, 2) preliminary questions, 3) software traceability definitions, 4) our proposed quality model, and 5) a debriefing. In the introduction, we briefly explained the purpose of the quality model and discussed the scope of the model. We also introduced the scales that were used throughout the questionnaire to ask the subject for its rating of each element of the quality model. The second part contained questions about the traceability expertise of a subject. In the third part, we provided illustrative diagrams to make the subject familiar with the preliminary assumptions from which the proposed TAM was derived. In the fourth part, we requested the subject to rate each element in order to evaluate the usefulness of the proposed model. Thereby, we requested the subject to rate each traceability gate's relevance on a scale between 1 (minor) and 5 (major), and to rate each traceability problem in terms of criticality on a scale between 1 (non-critical) and 5 (critical). In the last part, we asked the subject about traceability problems that are missing in the proposed quality model. We considered a subjects response as agreement to the completeness of our model, if no missing traceability problems were mentioned.

B. Study Results

Figure 8 compares the relevance ratings given by the 13 subjects for the traceability gates $\boxed{1}$ – $\boxed{6}$. All traceability gates were rated highly relevant by the subjects with a median of either 4 or 5. Individual statistics are as follows: $\boxed{1}$ (avg: 4.31, med: 5, std: 0.82), $\boxed{2}$ (avg: 4.00, med: 4, std: 0.68), $\boxed{3}$ (avg: 3.92, med: 4, std: 0.62), traceability gates $\boxed{4}$ (avg: 3.62, med: 4, std: 1.33), $\boxed{5}$ (avg: 3.69, med: 4, std: 0.99), and $\boxed{6}$ (avg: 3.46, med: 4, std: 0.93).

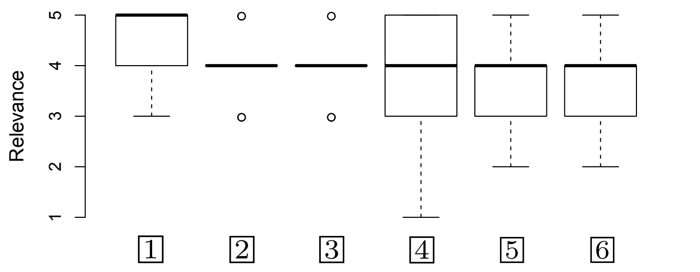


Fig. 8. Relevance ratings for the traceability gates $\boxed{1}$ – $\boxed{6}$ given by the 13 traceability experts

Figure 9 compares the criticality ratings given by the 13 subjects for the traceability problems $\boxed{1_M}$ – $\boxed{6_S}$. All traceability problems of the problem category missing traceability information were rated highly critical by the subjects with a median of 4 or 5 and individual statistics as follows: $\boxed{1_M}$ (avg: 4.62, med: 5, std: 0.84), $\boxed{2_M}$ (avg: 4.38, med: 5, std: 0.74), and $\boxed{3_M}$ (avg: 4.15, med: 4, std: 0.66). The traceability problems of the problem category missing traceability data were not rated homogeneously. While $\boxed{5_M}$ (avg: 3.69, med: 4, std: 0.99), and $\boxed{6_M}$ (avg: 3.46, med: 4, std: 1.01) were rated highly critical by the subjects with a median of 4; the traceability problem $\boxed{4_M}$ (avg: 3.23, med: 3, std: 1.19) was rated as moderately critical by the subjects with a median of 3 and an average of 3.23, moderately above the center value of the rating scale. All Traceability problems of the problem category superfluous traceability information were rated slightly critical by the subjects with a median of 2 or 3 and averages below the center value of the rating scale. The individual statistics per class are: $\boxed{1_S}$ (avg: 1.92, med: 2, std: 1.14), $\boxed{2_S}$ (avg: 2.12, med: 2, std: 0.95), and $\boxed{3_S}$ (avg: 2.38, med: 3, std: 0.74). The traceability problems of the problem category superfluous traceability data were rated moderately critical by the subjects with a median of 3 and averages moderately above the center value of the

[1]http://www.tu-ilmenau.de/sp/studien/tam_survey

rating scale. The individual statistics are: $\boxed{5_S}$ (avg: 3.15, med: 3, std: 1.23) and $\boxed{6_S}$ (avg: 3.08, med: 3, std: 1.27).

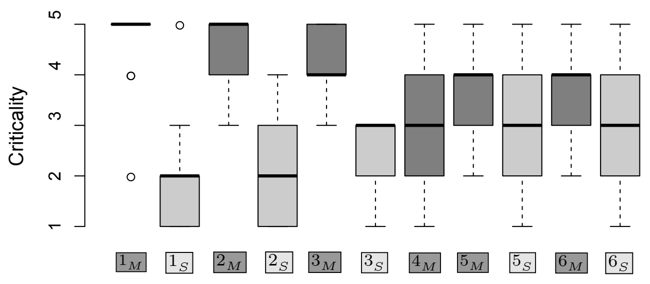


Fig. 9. Criticality ratings for the traceability problems $\boxed{1_M}$ – $\boxed{6_S}$ from the 13 traceability experts

Figure 10 summarizes the subjects' ratings per traceability entity. The horizontal alignment of a bubble represents the relevance of the associated traceability gate. The vertical alignment of a bubble represents the criticality of the associated traceability problem. Thereby, traceability problems of the problem category missing are shown in dark-gray and traceability problems of the problem category missing are shown in light-gray. The radius of each bubble represents the inverse standard deviation among the subjects' criticality ratings. That means the bigger the circle the higher the subjects' consensus.

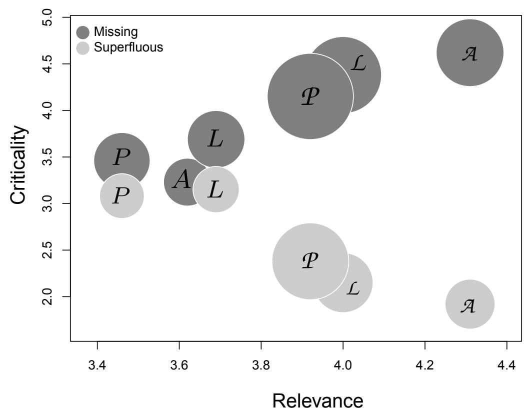


Fig. 10. Overview of traceability gate relevance and traceability problem criticality ratings per traceability entity, where the extent of each circle represents the subjects' rating consensus

Twelve subjects rated the proposed TAM as complete, and did not propose additional problem classes in the debriefing section. The remaining subject raised the following issue: "[..] maybe directions of trace links are not covered fully by your proposal? For example, bi-directional link obligated, but available only in one direction". Since trace link classes are directed in our proposed TAM, each trace link class consists of a source artifact class and a target artifact class (see Section II). A bi-directional traceability obligation between two artifact classes $\mathcal{A}_1$ and $\mathcal{A}_2$ would be mandated by the following two trace link classes: $\mathcal{A}_1 \rightarrow \mathcal{A}_2$ and $\mathcal{A}_2 \rightarrow \mathcal{A}_1$. If the TIM would specify only one direction, e.g. $\mathcal{A}_1 \rightarrow \mathcal{A}_2$, then a missing trace link class would be detected due to the criterion of $\boxed{2_M}$. Hence, we are confident that bi-directional links are in fact covered by the proposed model.

C. Threats to Validity

Quantitative feedback was collected from traceability experts through a questionnaire survey. A potential threat to the validity derives from the possibility that subjects can misunderstand the underlying concepts of the presented traceability problem classifications, which was addressed by the following countermeasure. We added a comprehensive briefing section to the questionnaires to illustrate the underlying concepts. The briefing section's effectiveness of ensuring the subject's conceptual understanding was extensively tested with pilot participants, who read the briefing and completed the questionnaire. Each pilot participant was interviewed. Based on these findings, we incrementally improved the briefing section of the questionnaire. Further, the relatively few study subjects are a potential threat to the study result conclusion validity. However, a subject matter experts survey is always a trade-off between the number of participants and the average level of expertise within the study, because the number of available experts is restricted. To assure a decent average level of expertise on the traceability topic, the study has been restricted to the group of 13 people.

V. Related Work

Systematic quality models are a well studied and established concept in the broader sense of software quality. The works of McCall et al. [21], [22], [23], Boehm [24], and Kitchenham et al. [25] as well as the ISO-25010:2011 standard [26] propose software characteristics for assessing software quality.

Similarly, specific quality models have been proposed with a focus on the specification and management of software requirements. Boehm [27] provides a taxonomy of satisfactory software requirements specification. A comprehensive overview and definitions of quality attributes in software requirements specification was given by Davis et al. [2]. These definitions found their way into IEEE's recommendations for requirement specification [3] as well as ISO's guidelines for applying requirements-related software engineering processes [4].

Furthermore, methodologies for systematically classifying faults and problems within software requirements specifications and software in general have been studied extensively and are well established today. Jackson [28], for example, proposed the definition of problem frames for defining the shape of a problem. Various studies employed similar methodologies to systematically define classifications of software anomalies [29], [30] and software requirement faults [31], [32].

Various researchers proposed traceability metrics to characterize traced software artifacts. For example, Pfleger and

Bohner [33] proposed software maintenance metrics for traceability graphs. They distinguish vertical traceability metrics (i.e., cyclomatic complexity and size) and horizontal traceability metrics. While vertical metrics are meant to characterize the developed product, horizontal metrics are meant to characterize the development process. To generically measure the complexity of requirements traceability, Costello et al. [34] proposed the use of linkage statistic metrics. Dick [35] extends the idea of analyzing traceability graphs by introducing trace link semantics, which he calls rich traceability. Main advantage of his approach is that propositional reasoning can be applied to analyze traceability relationships for their consistency. Instead of calculating traceability metrics, Canfora and Cerulo [36] employ information retrieval algorithms for the purpose of impact analysis. The authors leverage the description of requirements changes to automatically analyze the impact of a requirements change. Briand et al. [37] analyze traceability information between design and test data to automatically characterize design changes. In [38], we studied the applicability of traceability metrics for assessing the implementation risks of requirements.

Several authors conducted empirical research on requirements traceability and argue the need for planned traceability and defined traceability strategies. Gotel and Finkelstein argue in [39] that the knowledge about stakeholders that contributed to traced artifacts helps improving traceability. Ramesh identified two general groups of traceability users, which he refers to as low-end and high-end traceability users [40]. While low-ends users rely on simple dependencies among requirements, high-end users leverage much more sophisticated traceability schemes. Ramesh and Jarke conducted a large practitioner and tool study on traceability in [41], where they pointed out that traceability links should be strongly typed in order to avoid semantic misinterpretations. As a result, the authors proposed a traceability meta-model and reference models as guidance for practitioners. The use of a traceability information model as a necessary condition to employ traceability is advocated in [15]. Arkley and Riddle [42] conducted a case study on a software project, which successfully leveraged traceability. They concluded that the success of the observed traceability system was mainly influenced by two facts: (i) general traceability needs were examined to support project participants in their tasks and (ii) the traceability information model was systematically tailored to the identified needs.

Although, the approach of systematically defining classifications of software problems is well known, such a classification was lacking for the quality of requirements traceability data so far. Previous works on classifying problems in the area of requirements traceability, solely investigated on the correctness of approaches for automated traceability data retrieval [43], [44].

VI. Discussion and Future Work

Traceability provides an important means to support high impact development decisions in software projects. In practice however, traceability is planned, created, and maintained by humans, who make mistakes. Recent industrial case studies [6], [10], [12] showed that software projects suffer from a large number of traceability problems, which practitioners struggle to detect reliably. In this paper, we proposed a TAM providing guidance in systematically assessing software projects for potential traceability problems. For each traceability element, our TAM defines applicable assessment criteria.

We evaluated the proposed TAM by conducting a survey with 13 traceability experts. We found that all participants of our study considered the proposed assessment model to be complete. Further, the responses of the participants attested that the proposed TAM and its quality criteria are of high practical relevance. We also found that the criticality weighting of the traceability problem followed a pattern. Problems related to the quality attribute *completeness* are consistently rated as more important than problems related to the quality attribute *appropriateness*. We assume this pattern is attributed to the different problem implication. Problems related to the quality attribute *appropriateness* imply that unnecessary effort was spent. Problems related to the quality attributes *completeness* imply that traceability based decisions are made on incomplete data. Especially within the context of safety critical software, this can be a potential threat to the functional system safety. Comparing the exposures to safety risks and with monetary risks, a discrepancy of criticality is obvious.

The work reported in this paper has two important contributions to the requirements engineering community, in particular, to the work on trusted traceability. First, the proposed model provides clear guidance on how to systematically assess each traceability element in a software project for potential structural traceability problems. Therefore, the proposed TAM specifies unambiguous assessment criteria. Second, the results of our expert survey provide weights that quantify the criticality, and thus the potential impact of the traceability problems.

A. Limitations

The proposed TAM has two limitations, which we want to highlight to clearly define what can be expected from our model. First, our model cannot be used to identify semantically incorrect trace links between artifacts. For this purpose, a semantic analysis of the trace links and artifacts would be required, which is out of the scope of this work. Second, our model can only be used to identify incomplete or redundant traceability data. It cannot be used to identify whether or not the existing traceability in a project is semantically complete. Similarly, a semantic analysis would be required for this kind of assessment.

B. Future Research

Based on the proposed quality model, we want to provide tool support to automate the assessment of traceability. Future work could also focus on the semantic correctness and completeness of trace links to further support practitioner with traceability assessment. Information retrieval approaches or heuristics could be used to automate the semantic analysis of traceability, which today are mainly used for automated trace retrieval. However, the semantic analysis of development artifacts remains a challenging task.

Acknowledgment

We are supported by the German Ministry of Education and Research (BMBF) grant 01IS14026B.

References

[1] O. C. Gotel and A. C. Finkelstein, "An analysis of the requirements traceability problem," in *Requirements Engineering, 1994., Proceedings of the First International Conference on*. IEEE, 1994, pp. 94–101.

[2] A. Davis, S. Overmyer, K. Jordan, J. Caruso, F. Dandashi, A. Dinh, G. Kincaid, G. Ledeboer, P. Reynolds, P. Sitaram *et al.*, "Identifying and measuring quality in a software requirements specification," in *Software Metrics Symposium, 1993. Proceedings., First International*. IEEE, 1993, pp. 141–152.

[3] IEEE, "Ieee recommended practice for software requirements specifications," IEEE, Tech. Rep. 830-1998, 1998.

[4] ISO, IEC, and IEEE, "ISO/IEC/IEEE 29148:2011 – systems and software engineering – life cycle processes – requirements engineering," ISO/IEC/IEEE, Tech. Rep., 2011.

[5] J. Cleland-Huang, O. Gotel, J. Huffman Hayes, P. Mäder, and A. Zisman, "Software traceability: Trends and future directions," in *Proc. of the 36th International Conference on Software Engineering (ICSE), Hyderabad, India*, 2014.

[6] P. Rempel, P. Mäder, and T. Kuschke, "An empirical study on project-specific traceability strategies," in *Proceedings of the 21st IEEE International Requirements Engineering Conference (RE)*. IEEE, 2013, pp. 195–204.

[7] M. Zeller, K. Höfig, and M. Rothfelder, "Towards a cross-domain software safety assurance process for embedded systems," in *Computer Safety, Reliability, and Security*. Springer, 2014, pp. 396–400.

[8] T. P. Kelly, "Arguing safety - a systematic approach to managing safety cases," Ph.D. dissertation, University of York, 1999.

[9] P. Mäder, O. Gotel, and I. Philippow, "Motivation matters in the traceability trenches," in *Proceedings of the 17th IEEE International Requirements Engineering Conference*. IEEE, 2009, pp. 143–148.

[10] P. Mäder, P. L. Jones, Y. Zhang, and J. Cleland-Huang, "Strategic traceability for safety critical projects," *Software, IEEE*, vol. 30, no. 3, pp. 58–66, 2013.

[11] P. Rempel, P. Mäder, T. Kuschke, and I. Philippow, "Requirements traceability across organizational boundaries-a survey and taxonomy," in *Requirements Engineering: Foundation for Software Quality*. Springer, 2013, pp. 125–140.

[12] P. Rempel, P. Mäder, T. Kuschke, and J. Cleland-Huang, "Mind the gap: Assessing the conformance of software traceability to relevant guidelines," in *Proc. of the 36th International Conference on Software Engineering (ICSE), Hyderabad, India*, 2014.

[13] O. Gotel, J. Cleland-Huang, J. H. Hayes, A. Zisman, A. Egyed, P. Grünbacher, A. Dekhtyar, G. Antoniol, J. Maletic, and P. Mäder, "Traceability fundamentals," in *Software and Systems Traceability*. Springer, 2012, pp. 3–22.

[14] F. A. Pinheiro, "Requirements traceability," in *Perspectives on software requirements*. Springer, 2004, pp. 91–113.

[15] P. Mäder, O. Gotel, and I. Philippow, "Getting back to basics: Promoting the use of a traceability information model in practice," in *ICSE Workshop on Traceability in Emerging Forms of Software Engineering, 2009. TEFSE*. IEEE, 2009, pp. 21–25.

[16] I. Menzel, M. Mueller, A. Gross, and J. Doerr, "An experimental comparison regarding the completeness of functional requirements specifications," in *RE10*, vol. 25, no. 1, IEEE. Elsevier, 2010, pp. 15–24.

[17] M. Mirakhorli and J. Cleland-Huang, "Using tactic traceability information models to reduce the risk of architectural degradation during system maintenance," in *Software Maintenance (ICSM), 2011 27th IEEE International Conference on*. IEEE, 2011, pp. 123–132.

[18] A. Delater, N. Narayan, and B. Paech, "Tracing requirements and source code during software development," in *ICSEA 2012, The Seventh International Conference on Software Engineering Advances*, 2012, pp. 274–282.

[19] S. Nejati, M. Sabetzadeh, D. Falessi, L. Briand, and T. Coq, "A sysml-based approach to traceability management and design slicing in support of safety certification: Framework, tool support, and case studies," *Information and Software Technology*, vol. 54, no. 6, pp. 569–590, 2012.

[20] ISO, "ISO 9000:2005 – quality management systems. fundamentals and vocabulary," ISO, Tech. Rep., 2005.

[21] J. A. McCall, P. K. Richards, and G. F. Walters, "Factors in software quality. volume 1. concepts and definitions of software quality," DTIC Document, Tech. Rep., 1977.

[22] ——, "Factors in software quality. volume 2. metric data collection and validation," DTIC Document, Tech. Rep., 1977.

[23] ——, "Factors in software quality. volume 3. preliminary handbook on software quality for an acquisiton manager," DTIC Document, Tech. Rep., 1977.

[24] B. W. Boehm, J. R. Brown, H. Kaspar, M. Lipow, G. J. MacLeod, and M. J. Merrit, *Characteristics of software quality*. North-Holland Publishing Company, 1978, vol. 1.

[25] B. Kitchenham, S. Linkman, A. Pasquini, and V. Nanni, "The squid approach to defining a quality model," *Software Quality Journal*, vol. 6, no. 3, pp. 211–233, 1997.

[26] ISO, "ISO 25010:2011 – systems and software engineering – systems and software quality requirements and evaluation (SQuaRE) – system and software quality models," International Organization for Standardization, Geneva, Switzerland, ISO 25010:2011, 2011.

[27] B. W. Boehm, "Verifying and validating software requirements and design specifications," *Software, IEEE*, vol. 1, no. 1, pp. 75–88, 1984.

[28] M. Jackson, *Problem frames: analysing and structuring software development problems*. Addison-Wesley, 2001.

[29] R. B. Grady *et al.*, *Practical software metrics for project management and process improvement*. Prentice Hall Englewood Cliffs, 1992, vol. 3.

[30] R. Chillarege, "Orthogonal defect classification," *Handbook of Software Reliability Engineering*, pp. 359–399, 1996.

[31] J. H. Hayes, "Building a requirement fault taxonomy: Experiences from a nasa verification and validation research project," in *Software Reliability Engineering, 2003. ISSRE 2003. 14th International Symposium on*. IEEE, 2003, pp. 49–59.

[32] L. Miller, J. Hayes, and S. Mirsky, "Guidelines for the verification and validation of expert system software and conventional software, volume 1," US Nuclear Regulatory Commission, Tech. Rep., 1995.

[33] S. L. Pfleeger and S. A. Bohner, "A framework for software maintenance metrics," in *Software Maintenance, 1990., Proceedings., Conference on*. IEEE, 1990, pp. 320–327.

[34] R. J. Costello and D.-B. Liu, "Metrics for requirements engineering," *Journal of Systems and Software*, vol. 29, no. 1, pp. 39–63, 1995.

[35] J. Dick, "Rich traceability," in *Proceedings of the 1st international workshop on traceability in emerging forms of software engineering, Edinburgh, Scotland*, 2002, pp. 18–23.

[36] G. Canfora and L. Cerulo, "Impact analysis by mining software and change request repositories," in *Software Metrics, 2005. 11th IEEE International Symposium*. IEEE, 2005, pp. 9–pp.

[37] L. C. Briand, Y. Labiche, and G. Soccar, "Automating impact analysis and regression test selection based on uml designs," in *Software Maintenance, 2002. Proceedings. International Conference on*. IEEE, 2002, pp. 252–261.

[38] P. Rempel and P. Mäder, "Estimating the implementation risk of requirements in agile software development projects with traceability metrics," in *Requirements Engineering: Foundation for Software Quality*. Springer, 2015, pp. 125–140.

[39] O. Gotel and A. Finkelstein, "Extended requirements traceability: Results of an industrial case study," in *Proceedings 3rd IEEE International Symposium on Requirements Engineering*. IEEE, 1997, pp. 169–178.

[40] B. Ramesh, "Factors influencing requirements traceability practice," *Communications of the ACM*, vol. 41, no. 12, pp. 37–44, 1998.

[41] B. Ramesh and M. Jarke, "Toward reference models for requirements traceability," *IEEE Trans. Softw. Eng.*, vol. 27, no. 1, pp. 58–93, 2001.

[42] P. Arkley and S. Riddle, "Tailoring traceability information to business needs," in *Requirements Engineering, 14th IEEE International Conference*. IEEE, 2006, pp. 239–244.

[43] A. Egyed, S. Biffl, M. Heindl, and P. Grünbacher, "A value-based approach for understanding cost-benefit trade-offs during automated software traceability," in *Proc. of the 3rd workshop on TEFSE*. ACM, 2005, pp. 2–7.

[44] C. Ingram and S. Riddle, "Cost-benefits of traceability," in *Software and Systems Traceability*. Springer, 2012, pp. 23–42.