

A dissertation submitted to the
Fakultät für Informatik und Automatisierung
Technische Universität Ilmenau

Continuous Assessment of Software Traceability

for the degree of
DOKTOR-INGENIEUR (DR.-ING.)
by

Dipl.-Wirt.-Inf.
Patrick Rempel

born December 15, 1978
in Weimar, Germany

accepted on the recommendation of
Prof. Dr.-Ing. Patrick Mäder (JP), TU Ilmenau
Prof. Dr.-Ing. habil. Armin Zimmermann, TU Ilmenau
Prof. Dr. Barbara Paech, Universität Heidelberg

Submitted: June 29, 2015

Defended: February 16, 2016

urn:nbn:de:gbv:ilm1-2016000257

Acknowledgment

I would like to thank my professor, Prof. Dr. Ilka Philippow, who provided me the great opportunity to work and research at the Technische Universität Ilmenau. I am also very thankful to Dr. Patrick Mäder for supervising my research work over the last three years. His great passion for research in general, as well as for software engineering and traceability in particular, was an enduring source of inspiration and motivation for me. Furthermore, I would like to thank Prof. Dr. Barbara Paech for examining my work and providing valuable feedback on an early version of the thesis. I also thank Prof. Dr. Armin Zimmermann for examining my work.

I was funded for the last three years by the German Ministry of Education and Research (BMBF): grants 16V0116 and 01IS14026B.

I want to thank all members of the Software Systems/Process Informatics Group at the Technische Universität Ilmenau. Thanks to all my colleagues for providing such a pleasant environment and valuable feedback on my research: Tobias Kuschke, Elke Bouillon, Theodora Kickova, Steffen Lehnert, Qurat-Ul-Ann Farooq, Oswald Kowalsky, Jana Wäldchen, Marco Seeland, Nedal Alaqraa, Stefan Wendler, and Nils Würfel. In memoriam, I would like to thank Heiner Kotula for his technical and non-technical support.

I would like to thank all participants of the empirical studies, which I conducted throughout the last three years. Each of them spent a lot of time to share their deep expertise in personal interviews or by filling comprehensive questionnaires. The research presented in this thesis would have been impossible without the contributions of all these experts.

My special thanks go out to my entire family. My parents provided support at any given opportunity. My sister helped me with difficult decisions. Finally, and most deeply, I thank Magdalena, Katharina, and Elisabeth for being so patient and modest.

Abstract

Traceability is a critical element of any rigorous software development process. It is required by numerous software lifecycle activities such as, for example, safety analysis, change impact analysis, coverage analysis, and compliance verification. Safety guidelines such as ISO 61508 and its domain specific derivatives explicitly require the implementation of software traceability.

Although the crucial importance of traceability is commonly acknowledged, software development projects rarely follow explicit traceability strategies. Traceability is rarely planned or systematically created but should rather be regarded as a desultory ad-hoc effort. In result, existing traces are potentially of dubious quality but serve as the foundation for high impact development decisions. To ensure that traceability is trustworthy, the fitness for purpose of a project's traceability implementation must be thoroughly ascertained, especially within the context of safety-critical software. Assessing the fitness for purpose is an intricate problem for several reasons. Depending on the project specific traceability goals, different ways of traceability are applied within multiple projects. The development of safety-critical software is subject to different regulations with diverse provisions that need to be regarded.

This thesis will present an approach to systematically assess the fitness for purpose of a project's traceability implementation, comprising two parts. The first part supports the planning of purposed traceability, which is a prerequisite for the traceability assessment. Based on the planning results, the second part supports the actual assessments. It defines an analytical traceability assessment model. This model provides a comprehensive classification of possible traceability problems and defines assessment criteria to systematically detect these problems.

The results of a traceability experts survey suggest that proposed traceability problem classification is complete and defines relevant assessment criteria. The proposed assessment approach was applied in two studies. The study results indicate that the proposed assessment provides support for multiple purposes. It can be used in order to determine the feasibility of important software lifecycle activities and the cost effectiveness of a project's traceability implementation. Safety-critical software projects can be supported with their safety argument. The compliance of projects' traceability implementations to safety guidelines can be determined.

Zusammenfassung

Die Nachvollziehbarkeit von Anforderungen ist wichtiges Qualitätsmerkmal der Softwareentwicklung. Für eine Vielzahl von Softwareentwicklungsaktivitäten ist die Nachvollziehbarkeit von Anforderungen eine notwendige Voraussetzung. Dazu gehören unter anderem die Analyse funktionaler Sicherheit, die Einflussanalyse, die Analyse des Abdeckungsgrades oder die Compliance. Für die Entwicklung sicherheitskritischer Softwaresysteme ist dieses Qualitätsmerkmal von besonderer Bedeutung. Daher wird dieses von entsprechenden Richtlinien zur Entwicklung sicherheitskritischer Software explizit vorgeschrieben.

Obwohl die Relevanz der Nachvollziehbarkeit in Softwareprojekten allgemein bekannt ist, findet nur in wenigen Fällen eine systematische Planung zur Erreichung dieses Qualitätsmerkmals Anwendung. Häufig wird Nachvollziehbarkeit erst nachträglich umgesetzt. Daraus resultieren oft unvollständige Implementierungen der Nachvollziehbarkeit, die trotzdem als Grundlage für schwerwiegende Entscheidungen herangezogen werden. Aus diesem Grunde sollten die entsprechenden Implementierungen einer eingehenden Prüfung unterzogen werden, besonders im Rahmen der Entwicklung sicherheitskritischer Systeme. Dazu sind jedoch eine Vielzahl von Herausforderungen zu meistern. Zum einen hängt die Nachvollziehbarkeit von den projektspezifischen Zielen ab. Bei sicherheitskritischen Systemen müssen oft Vorgaben aus Richtlinien erfüllt werden. Auch die Nutzung der Nachvollziehbarkeit ist sehr stark von den jeweiligen Zielen abhängig.

In dieser Arbeit wird ein Ansatz zur systematischen Prüfung von Softwareprojekten im Hinblick auf deren Nachvollziehbarkeit der Anforderungen vorgeschlagen. Eine notwendige Voraussetzung für den Prüfansatz ist die präzise Planung und Definition der Nachvollziehbarkeit von Anforderungen in einem Softwareprojekt. Daher wird im Rahmen dieser Arbeit ein entsprechender Planungsansatz präsentiert. Weiterhin wird ein analytisches Modell zur systematischen Prüfung der Nachvollziehbarkeit in Softwareprojekten präsentiert. Dieses Modell umfasst eine vollständige Klassifikation möglicher Fehlertypen. Außerdem werden Kriterien zur systematischen Erkennung dieser Fehler vorgeschlagen.

Die Ergebnisse einer Expertenbefragung bestätigen die Vollständigkeit des analytischen Prüfmodells. Zudem wurde der vorgeschlagene Ansatz zur systematischen Prüfung der Nachvollziehbarkeit von Anforderungen in zwei Studien evaluiert. Dabei konnte der Nutzen des Ansatzes für die Entwicklung von sicherheitskritischer und nicht sicherheitskritischer Software nachgewiesen werden.

Contents

1. Introduction	1
1.1. Motivation	2
1.2. Contributions	3
1.3. Thesis Outline	4
2. Software Traceability	7
2.1. Fundamentals	7
2.2. Traceability Characteristics	9
2.3. Traceability Lifecycle	11
2.4. Traceability Research Issues	12
2.5. State of the Art	13
2.5.1. Empirical Work on Traceability Problems	14
2.5.2. Definitional Approaches	15
2.5.3. Anticipatory Approaches	15
2.5.4. Analytical Approaches	16
2.6. Critique of the State of the Art	18
3. The Traceability Assessment Approach	21
3.1. Traceability Assessment Challenges	21
3.1.1. Purposed	21
3.1.2. Trusted	22
3.1.3. Automation	23
3.2. Characterizing Traceability Implementations	23
3.3. Overview of the Assessment Approach	26
3.4. Usage Scenarios of Traceability Assessment	27
4. Planning for Purposed Traceability	31
4.1. Identifying Traceability Requirements	31
4.1.1. A Model for Traceability Requirements	32
4.1.2. Identifying Software Lifecycle Related Goals	35
4.1.3. Identifying Goal Specific Activities	36
4.1.4. Identifying Goals that Require Traceability	37
4.1.5. Identifying Traceability Implementation Activities	38

4.2.	Identifying Required Traceability Information	39
4.2.1.	A Model for Required Traceability Information	40
4.2.2.	Identifying Required Trace Path Types	42
4.2.3.	Identifying Required Trace Link Types	44
4.3.	Justifying the Purpose of Required Trace Link Types	45
4.4.	Summary	47
5.	Assessing the Fitness for Purpose of Implemented Traceability	49
5.1.	A Traceability Assessment Model	50
5.2.	Quality Attributes of a Purposed Traceability Implementation	52
5.3.	Assessable Traceability Implementation Properties with Respect to Purposed Traceability	53
5.4.	Traceability Problems	57
5.4.1.	Problems Related to the Completeness	57
5.4.2.	Problems Related to the Appropriateness	60
5.4.3.	Problems Related to the Correctness	64
5.5.	Dependencies Among the Traceability Problems	67
5.6.	Performing a Traceability Assessment	70
5.6.1.	Step 1: Collecting Traceability Implementation Data	70
5.6.2.	Step 2: Extracting Types from Implemented Traceability Data	72
5.6.3.	Step 3: Mapping Implemented Traceability Data to Required Traceability Information	74
5.6.4.	Step 4: Assessing the Implemented Traceability Data	74
5.7.	Summary	77
6.	Tool Support for Continuous Traceability Assessment	79
6.1.	The Purpose Induced Software Traceability Assessor (PurISTA) Prototype	79
6.1.1.	Traceability Store	80
6.1.2.	Traceability Planner	82
6.1.3.	Traceability Collector	82
6.1.4.	Traceability Browser	84
6.1.5.	Traceability Assessor	85
6.2.	Summary	87
7.	Evaluation	89
7.1.	Research Questions	89
7.2.	Study 1: Traceability Assessment Model	91
7.3.	Study 2: Value Driven Traceability Implementations	93
7.4.	Study 3: Regulated Traceability Implementations	97
7.5.	Study 4: Traceability Assessment Results	101

7.6.	Discussion	106
7.6.1.	Research Question 1: Relevance	106
7.6.2.	Research Question 2: Completeness	107
7.6.3.	Research Question 3: Feasibility of Software Lifecycle Activities	108
7.6.4.	Research Question 4: Cost-effective Implementation	109
7.6.5.	Research Question 5: Compliance	109
7.6.6.	Research Question 6: Migration	110
7.6.7.	Research Question 7: Continuous Assessment	110
7.6.8.	Limitations	110
7.7.	Threats to Validity	111
7.7.1.	Construct Validity	112
7.7.2.	External Validity	112
7.7.3.	Internal Validity	112
7.7.4.	Reliability	113
8.	Conclusions and Outlook	115
8.1.	Summary	115
8.2.	Future Work	116
	List of Figures	119
	List of Tables	123
	Bibliography	125
A.	Evaluation Material of Study 1	137
B.	Evaluation Material of Study 2	141

1. Introduction

The emerging Internet of Things (IoT) paradigm is advocating the interconnectedness of devices by applying the interaction model of the World Wide Web. Analysts forecast that the number of IoT devices will grow to 26 billion units in 2020 [Middleton et al. 2013]. The global embedded systems market, which was valued at USD 140.32 billion in 2013, is expected to grow at a Compound Annual Growth Rate (CAGR) of 6.3% to USD 214.39 billion in 2020. The segment of embedded software is even expected to grow at a CAGR of 8.1% between 2014 and 2020 [Grand View Research 2014]. Analysts also forecast that developing embedded software for safety critical systems will become increasingly important over the next years [Grand View Research 2014].

One important aspect of developing safety critical software is the compilation of a safety case to argue that the developed system is safe for use [Zeller et al. 2014]. The safety argument needs to demonstrate that all safety risks have been identified and how exactly they have been mitigated. In practice, traceability is considered an important mean to support those safety arguments [Kelly 1999], because it provides the “*ability to describe and follow the life of a requirement in both forwards and backwards direction*” [Gotel and C. Finkelstein 1994]. This ability can be used to efficiently demonstrate that all safety requirements have been validated, satisfied, and realized, and that their origin is documented [Rierson 2013].

Besides supporting the safety argument for the development of safety-critical software, traceability is considered a “*critical element of any rigorous software development process*” [COEST 2015]. It provides support for numerous software engineering activities such as, for example, change impact analysis, coverage analysis, and compliance verification. Change impact analysis activities are essential for the software change process. The analysis results can be used to determine the software artifacts that are impacted by a planned change [von Kethen 2002]. Furthermore, selecting relevant test cases for regression tests can effectively be supported [Briand et al. 2002]. Coverage analysis activities provide insights on the completion status of a software system under development. For example, the analysis results can be used to determine whether or not a specific requirement artifact is covered by appropriate test artifacts [Lormans and van Deursen 2005]. It is also used to monitor a system’s overall requirements coverage [Kirova et al. 2008]. Activities to verify the compliance of software artifacts with regulatory codes improve the accountability of the developed system [Breux et al. 2006; Cleland-Huang et al. 2010].

1. Introduction

The fact that traceability provides several effective means to support the safety argument for safety-critical software systems, shows its high importance for the development of embedded systems. However, the variety of software engineering activities, where traceability is useful for the quality of the developed system, shows that its relevance is not limited to the safety domain. This practical relevance is also reflected by the fact that traceability is explicitly demanded by safety guidelines [IEC 61508:2010; DO-178C; ISO 26262-6:2011; ECSS 2009], software quality models [McCall et al. 1977; Davis et al. 1993], software engineering standards [ISO/IEC/IEEE 12207:2008; ISO/IEC/IEEE 29148:2011], and software development maturity frameworks [ISO/IEC 15504:2004; CMMI-DEV 2006].

1.1. Motivation

Although the crucial importance of traceability is commonly acknowledged, software development projects rarely follow explicit traceability strategies [Mäder et al. 2009b]. Instead, traceability in current software development practices is rarely planned or systematically created but rather a desultory ad-hoc effort and often implemented as an afterthought [Rempel et al. 2013; Mäder et al. 2013]. Besides, traceability is mostly implemented by humans who make mistakes that often remain undetected [Hayes and Dekhtyar 2005; Regan et al. 2012]. Leading to situations where implemented traceability is far away from being suitability for its originally intended purpose [Mäder et al. 2013].

To mitigate this lack of appropriate traceability practices, software engineering guidelines were developed by standardization organizations and certification authorities. General purpose software development guidelines (e.g., [CMMI-DEV 2006; ISO/IEC/IEEE 12207:2008; ISO/IEC/IEEE 29148:2011]) as well as guidelines for the development of safety-critical systems (e.g., [IEC 61508:2010]) provide traceability recommendations for practitioners. Some safety domains even developed domain-specific guidelines (e.g., avionics: [DO-178C], space: [ECSS 2009], automotive: [ISO 26262-6:2011], railway: [CENELEC 2011], medical: [FDA 2002]) to address their particular needs. Although these guidelines were created to support practitioners, organizations in practice struggle to implement accurate and complete sets of trace links [Rempel et al. 2013; Mäder et al. 2013; Rempel et al. 2014]. An analysis of the traceability information submitted by various organizations to the US Food and Drug Administration (FDA) as part of the medical device approval process in the United States showed a significant traceability gap between the traceability expectations as laid out in the FDA’s “Guidance for the Content of Premarket Submissions for Software Contained in Medical Devices” [FDA 2002], and the traceability data documented in the submissions [Mäder et al. 2013]. While all submissions attempted to satisfy the FDA’s traceability guidelines, serious deficiencies were found in almost all the submissions in terms of missing traceability

paths, missing and redundant trace links, and problems in trace granularity, which made it very difficult to understand the rationale for individual links. The criticality of software engineering activities (e.g., safety argument, change impact analysis) that are conducted in practice with incomplete traceability data gives cause for concerns.

All these observations in current industrial practice suggest that there is a gap between required and implemented traceability. This traceability gap demands for adequate traceability assessments to detect these gaps [Merilinna and Pärssinen 2010]. Researchers argue that traceability must be purposed and trusted [Cleland-Huang et al. 2014]. However, they also stated that these goals remain a challenge, which has not yet been achieved. Since traceability is subject of gradual decay with the evolution of a software system, the fitness for purpose of a project’s traceability implementation needs to be assessed in a continuous and timely manner. In fact, the European Open-DO initiative [Comar et al. 2009], actively seeks to address the Big Freeze problem in which the significant cost and effort of the assessment and certification process makes it difficult to introduce change once the product is certified. The initiative urges for the integration of continuous assessments into the development process of safety-critical software.

It can be concluded that current practice lacks effective methods to assess the quality of traceability implementations. More systematic manners to detect existing existing traceability gaps are yet to be found. Further, the Big Freeze problem demands for automated techniques, allowing traceability assessments in a continuous manner.

1.2. Contributions

To address the lack of effective traceability assessment methods in today’s practice, a novel traceability assessment approach will be presented in this thesis. Therefore, five main contributions will be made. Each of the contributions will be separately discussed in the following:

1. An analytical **Traceability Assessment Model (TAM)**, essential to the proposed assessment approach, will be presented in this thesis. It provides means to assess a project’s traceability implementation for its fitness for purpose. The model defines quality attributes that are relevant for a traceability implementation’s fitness for purpose. Further, the TAM provides generalized definitions, how these abstract quality attributes can be applied to a concrete traceability data of a software system.
2. A comprehensive **classification of atomic traceability problems** will be provided as an integral part of the analytical TAM. Each atomic problem represents a specific type of shortcomings of traceability implementations that indicate the existence of a traceability gap. For each problem, generalized assessment rules will

1. Introduction

be presented. These rules are abstracted from concrete rule languages and development project specific concepts to ensure universal applicability of the assessment rules.

3. A **traceability assessment tool** was prototypically implemented to address the Big Freeze problem. The prototype allows the automatic execution of traceability assessments. It enables scenarios where the assessment of a project's traceability implementation is triggered by changes to software artifacts in a continuous manner.
4. Traceability requirements can vary, depending on the project specific goals. Therefore, a clear specification of a project's traceability requirements is a prerequisite for assessing the fitness for purpose of its traceability implementation. This thesis will present a **purpose-oriented traceability planning approach** to support the specification of traceability requirements. The output of the proposed planning approach is precisely defined by a meta-model.
5. The presented assessment approach has been **qualitatively and quantitatively evaluated** for its industrial applicability. Extensive interviews have been conducted with subjects from 17 industry partners. Feedback has been collected from 12 traceability experts on the TAM and its traceability problem classification. The automated assessment approach was applied to the development artifacts of four safety-critical software projects. Feedback has been collected from 17 safety-project participants and safety certifiers on the produced assessment results.

1.3. Thesis Outline

The remainder of this thesis is organized as follows:

Chapter 2: Software Traceability. This chapter sets the context of this thesis and establishes the research baseline. Fundamental elements and concepts of software traceability are introduced. The current state of the art is discussed with respect to the goals *trusted* and *purposed*. Finally, a critique of the current state of the art is provided, which identifies shortcomings of existing traceability approaches.

Chapter 3: The Traceability Assessment Approach. This chapter outlines a novel traceability assessment approach to remedy the identified shortcomings of existing approaches. Explicit challenges are derived from the critique of the state of the art. Additionally, general characteristics of traceability implementation data are discussed, which are relevant for the traceability assessment approach. Further, possible usage scenarios are envisioned to motivate the relevance of the proposed assessment approach.

The proposed assessment approach consists of two parts. Each part is introduced in a separate chapter.

Chapter 4: Planning for Purposed Traceability. This chapter presents the first part of the proposed traceability assessment approach. To assess the fitness for purpose of a project's traceability implementation, a precise definition of the required traceability information is needed. Therefore, a goal-oriented planning approach is presented, which derives a definition of required traceability information from project specific traceability goals.

Chapter 5: Assessing the Fitness for Purpose of Implemented Traceability. This chapter presents the second and core part of the traceability assessment approach. Relevant quality attributes are identified and defined to characterize the abstract fitness for purpose concept and to specify assessable properties. Further, a comprehensive traceability problem classification is provided.

Chapter 6: Tool Support for Continuous Traceability Assessment. This chapter describes to a prototypical tool implementation that supports the proposed traceability assessment approach. Main purpose of this prototype is to support the automatic execution of traceability assessment in a continuous and timely manner.

Chapter 7: Evaluation. This chapter discusses three studies that have been conducted to evaluate the proposed traceability assessment approach. First, in an interview study with 17 software companies, the applicability of the first part of the assessment approach was evaluated. Second, a questionnaire study with 13 traceability experts has been conducted to evaluate the traceability problem classification with respect to its completeness and level of importance. Third, a case study with four safety critical software projects has been conducted to evaluate the applicability of the proposed assessment approach. Required traceability was derived from three different safety standards through the proposed traceability planning approach. For the case study results, additional qualitative feedback has been collected from 17 safety project participants and certifiers. A discussion of potential threats to validity is provided and how these threats were mitigated.

Chapter 8: Conclusions and Outlook. This chapter concludes this thesis by summarizing its important findings and contributions. Based on the results of this thesis, possible future work is outlined.

Appendix A: Evaluation Material of Study 1. This appendix provides the evaluation material that was created for the second study.

1. Introduction

Appendix B: Evaluation Material of Study 2. This appendix provides the evaluation material that was created to collect qualitative feedback from certifiers and project participants in the third study.

2. Software Traceability

Traceability is an essential quality within the context of software system development. Already in the late seventies of the last century, software quality models emphasized the importance of traceability for any rigorous software development process [McCall et al. 1977]. One reason for its importance is the variety of software engineering activities, which are supported by traceability. It supports software engineering activities such as, for example but not limited to, safety analysis, change impact analysis, compliance verification, and coverage analysis. Although, the importance of traceability was recognized so long time ago, it is still subject of extensive research aiming to improve existing challenges that have not yet been solved satisfactorily [Cleland-Huang et al. 2014].

To summarize the context of this thesis, traceability related concepts and the state of the art are presented in this chapter. Section 2.1 defines fundamental traceability elements. Section 2.2 discusses important traceability dimensions. In Section 2.3, an overview of the traceability lifecycle is provided. Traceability research issues are discussed in Section 2.4. Section 2.6 provides a critical discussion of the existing state of the art and identifies problems of the existing approaches that need to be addressed.

2.1. Fundamentals

Traceability is at the most fundamental level the “*potential to relate data that is stored within artifacts of some kinds, along with the ability to examine this relationship*” [Gotel et al. 2012b]. Originally, traceability was defined 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 all periods of on-going refinement and iteration in any of these phases)” [Gotel and C. Finkelstein 1994]. This definition has a very requirements centric perspective. Hence, the more general traceability definition is used in this thesis, which is provided in [COEST 2015a]. The authors define traceability in the following way:

Definition 1 (Software traceability). *The ability to interrelate any uniquely identifiable software engineering artifact to any other, maintain required links over time, and use the resulting network to answer questions of both, the software product and its development process.*

2. Software Traceability

According to this definition, the following two main building blocks of traceability can be derived: *artifact* and *trace link*. The term artifact refers to any work product that is created throughout the software lifecycle. Within the context of the Rational Unified Process (RUP), artifacts are defined as “*a piece of information that is produced, modified, or used by a process*” [Kroll and Kruchten 2003]. Since this definition has a process centric perspective, the following more general definition of the term artifact is provided for this thesis:

Definition 2 (Artifact). *An artifact refers to any output, whether final or not, that is produced or maintained by any activity throughout the entire software lifecycle.*

As described above, the term trace link is the second building block of traceability. It refers to a concept that interrelates two artifacts. As defined in [Gotel et al. 2012b], a trace link is a “*single association forged between two [...] artifacts, one comprising the source artifact and one comprising the target artifact*”. The definition implies that a trace link has a direction indicating from where (origin) to where (destination) the trace link is established [Cleland-Huang et al. 2003].

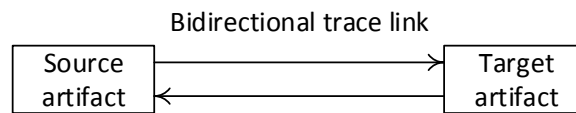


Figure 2.1.: Directed trace link between two artifacts, a source and a target artifact, that supports bi-directional traversal (adapted from [Gotel et al. 2012b])

As exemplified in Figure 2.1, the artifact from which the depicted trace link is originated holds the role source, and the artifact, to which the trace link directs, holds the role target. Explicitly stating a source and target of a trace link ensures that the semantics of the directionality are clear [Gotel et al. 2012b]. However, traceability is commonly considered to be bidirectional [Gotel and C. Finkelstein 1994], which means that a trace link can potentially be traversed in both forward and reverse direction [Wieringa 1995]. In consideration of these aspects, the term trace link is defined in this thesis as follows:

Definition 3 (Trace Link). *A trace link is a directed association established between two artifacts, the originating artifact that holds the role source and the destination artifact that holds the role target. A trace link can be traversed in both, forward and reverse direction.*

Traceability is established through the creation of trace links. An artifact is traceable if it is associated to one or many other artifacts via a trace link. Trace links typically form

a traceability network, as emphasized in the software traceability definition (see Definition 1). The Oxford Dictionary [Stevenson 2010] defines a network as “*an arrangement of intersecting horizontal and vertical lines*” and “*a group or system of interconnected people or things*”. Within the context of traceability, it is a common approach to represent networks of trace links as a traceability graph [S. Pfleeger and S. Bohner 1990; S. A. Bohner 1995; F. Pinheiro and Goguen 1996; Ramesh 1997; Ramesh and Jarke 2001; Egyed 2001; Vanhooft et al. 2007; Cleland-Huang et al. 2009; J. I. Maletic and Collard 2009; Schwarz et al. 2010]. Accordingly, the concept of a traceability graph is defined as follows for this thesis:

Definition 4 (Traceability Graph). *A traceability graph models and formally describes the system of traceable artifacts that are interconnected through trace links. The traceability graph consists of a set of vertices and a set of edges. A vertex of the traceability graph represents a traceable artifact. An edge of the traceability graph represents a trace link.*

A fundamental concept of graph theory are paths. A path is a finite non-null sequence, whose terms are alternately vertices and edges [Bondy and Murty 1976]. The edges and the vertices of a path are distinct. The length of a path is represented by the number of its edges, which is larger or equal than one. Accordingly, a specific trace path for a traceability graph is defined as follows:

Definition 5 (Trace Path). *A trace path is a relation between two traceable artifacts, the originating artifact that holds the role source, and the destination artifact that holds the role target. Thereby, the trace path consists of a sequence one to many distinct trace links such that any pair of adjacent trace links within the sequence is associated to one common artifact.*

2.2. Traceability Characteristics

Depending on the types of traceable artifacts that are related through a trace link, vertical and horizontal traceability relationships are distinguished in literature [S. Pfleeger and S. Bohner 1990; Lindvall and Sandahl 1996; Pohl 2010]. Vertical traceability refers to trace links between artifacts of the same type such as, for example, a trace link between two requirement artifacts. Horizontal traceability refers to trace links between artifacts of different types such as, for example, a trace link between a requirement and a design artifact.

Specifying a trace link’s source and target ensures that the semantics of the directionality are clear [Gotel et al. 2012b]. However, researchers argue that trace links should be usable bidirectional [Ramesh and Edwards 1992; Gotel and C. Finkelstein 1994]. Bidirectional traceability describes the ability to follow a trace link in both, forward

2. Software Traceability

and backward directions. Accordingly, forward traceability refers to following a trace link from the source to the target artifact. Backward traceability refers to the opposite.

Another dimension of traceability is the representation of trace links. In particular, traceability can either be represented explicitly or implicitly. Explicit traceability refers to a persistent storage of trace links. Over the years, researchers and industrial practice have proposed and tested a variety of trace link representation. Commonly used representations of explicit traceability are the following.

- *References.* Trace links are created through textual references [Lindvall and Sandahl 1996]. Thereby, the source artifact is annotated with a text that refers to a target artifact. These textual references are typically used in textual documents. A drawback of textual references is that these trace links can only be used bidirectionally if both, the source and the target are annotated with textual references.
- *Hyperlinks.* Trace links are created through hypertext references [Ebner and Kaindl 2002]. It better supports traceability users with the navigation between traceable artifacts.
- *Traceability matrix* - each row and each column of the matrix represents a traceable artifact [Jönsson and Lindvall 2005]. Trace links are created by putting a mark on the intersect of two traceable artifacts. Thus, traceability matrices unify the source and the target artifacts of all trace links in one holistic view and enable bidirectional traceability usage [Watkins and Neal 1994].
- *Traceability repositories.* Researchers argue that traceability should be implemented by means of an explicit registration of the artifacts and their links in a traceability repository [F. A. C. Pinheiro 2004]. These traceability repositories enable a holistic view to the complete software lifecycle [Mäder et al. 2008]. However, especially in projects with a heterogeneous artifact and tool landscape, the vision of a unified traceability repository requires enormous effort [Arkley et al. 2002].

Implicit traceability refers to relations that are not explicitly materialized through trace links. Thereby, implicit knowledge about artifact dependencies is used. Using implicit traceability reduces the number of trace links that needs to be created explicitly. In the past, different sources were used for mining implicit traceability relations [Lindvall and Sandahl 1996; Bianchi et al. 2000; Mäder et al. 2007]. The authors leveraged name tracing, system or domain knowledge, and structural dependencies.

- *Name tracing.* It derives implicit relations from artifacts with similar names. This approach assumes a coherent artifact naming strategy and its consistent implementation.

- *System or domain knowledge.* Experienced software engineers can use their expert knowledge about the system or domain, in order to infer artifact relations, which are not specified explicitly. The downside of this approach is that the knowledge cannot easily be transferred to other stakeholders.
- *Structural dependencies.* It derives implicit relations from artifact model definitions that specify structural dependencies. Definitions of the Rational Unified Process (RUP), for example, were used to infer implicit trace links in [Mäder et al. 2007].

It is important to notice that the characteristics of traceability vary across multiple projects, depending on factors such as the implemented software process, used tools and technologies, and the software lifecycle orchestration.

2.3. Traceability Lifecycle

A software project's traceability implementation, materialized as a traceability graph, runs through a traceability lifecycle, which is embedded within the comprehensive software lifecycle. In the past, various models were suggested to classify the traceability lifecycle activities [von Knethen and Paech 2002; F. A. C. Pinheiro 2004; Heindl and Biffel 2006; Mäder 2010; Winkler and von Pilgrim 2010; Gotel et al. 2012b; Cleland-Huang et al. 2014]. The authors distinguish the three activities traceability planning, traceability implementation, and traceability usage.

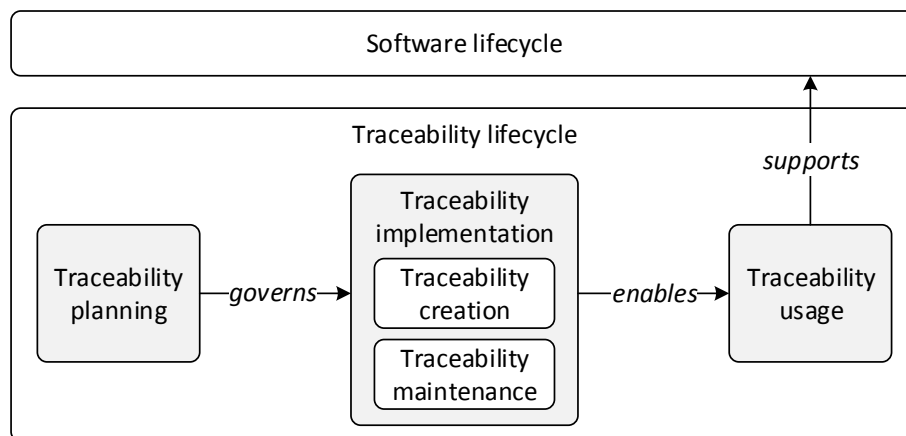


Figure 2.2.: Overview of the traceability lifecycle activities and their sub-activities

1. *Traceability planning.* It refers to the activity where a project specific traceability implementation is strategically planned. Initially, the stakeholders' traceability

2. Software Traceability

related needs are identified, because “*determining needs [...] is a precursor to any discussion about trace artifacts, trace links and mechanism*“ [Gotel et al. 2012a].

2. *Traceability implementation*. It refers to the activity where trace links are implemented to establish a traceability graph. The traceability implementation consists of two sub-activities *traceability creation* and *traceability maintenance*. The first sub-activity is concerned with creating trace links to produce traceable artifacts. The second sub-activity is concerned with maintaining existing trace links to keep the traceability graph updated, in accordance with the ongoing software evolution.
3. *Traceability usage*. It refers to the activity where the implemented trace links are used to answer questions related to the software and its development process.

Figure 2.2 provides an overview of the three traceability lifecycle activities and their sub-activities. The labeled arrows denote dependencies between the lifecycle activities. The result of the traceability planning activity governs the traceability implementation activity. The established trace links, which are the result of the traceability implementation activity, are a prerequisite for the traceability usage activity. Conducting the traceability usage activity supports and enables a variety of software lifecycle activities such as, for example, safety analysis [Panesar-Walawege et al. 2010; Cleland-Huang et al. 2012; Briand et al. 2014], change impact analysis [Bohner 1996; von Knethen 2002; Briand et al. 2002], coverage analysis [Lormans and van Deursen 2005; Kirova et al. 2008], and compliance verification [Breaux et al. 2006; Cleland-Huang et al. 2010].

2.4. Traceability Research Issues

The previous sections of this chapter have introduced and discussed essential aspects of software traceability. Over the past decades, these aspects have been subject to extensive research. Reasonable effort has been made to support the activities of the traceability lifecycle. However, some traceability research issues still remain, as they have not yet been solved satisfactorily. This section summarizes these remaining issues.

In 2014, the international conference on software engineering featured a “Future of Software Engineering (FOSE)” track, where leading software traceability experts were invited to report on open traceability research issues [Cleland-Huang et al. 2014]. The authors identified 8 desired traceability qualities and specified related goals, which are summarized in Table 2.1. The list of the not yet achieved goals emphasizes the great demand for further research in traceability area. As discussed in Section 1.1, several organizations struggle to implement adequate traceability that is fit-for-purpose. In current practice, there is a gap between the required and the implemented traceability. Traceability implementations suffer from serious deficiencies such as, for example, missing or redundant trace links, undermining their trustworthiness and damaging their

Table 2.1.: Overview of desired traceability qualities and their related goals [Cleland-Huang et al. 2014]

Quality	Goal
Purposed	Traceability is fit-for-purpose and supports stakeholder needs (i.e., traceability is requirements-driven).
Cost-effective	The return on investment (ROI) from using traceability is adequate in relation to the outlay of establishing it.
Configurable	Traceability is established as specified, moment-to-moment, and accommodates changing stakeholder needs.
Trusted	All stakeholders have full confidence in the traceability, as it is created and maintained in the face of inconsistency, omissions and change; all stakeholders can and do depend upon the traceability provided.
Scalable	Varying types of artifacts can be traced, at variable levels of granularity and in quantity, as the traceability extends through-life and across organizational and business boundaries.
Portable	Traceability is exchanged, merged and reused across projects, organizations, domains, product lines and supporting tools.
Valued	Traceability is a strategic priority valued by all; every stakeholder has a role to play and actively discharges his or her responsibilities.
Ubiquitous	Traceability is always there, without ever having to think about getting it there, as it is built into the engineering process; traceability has effectively “disappeared without a trace”.

reputation as an effective means to support critical software engineering activities. The focus of this thesis is therefore set on the traceability qualities *purposed* and *trusted*.

2.5. State of the Art

Although the goals related to purposed and trusted traceability may not have been achieved yet, traceability has been subject to extensive research over the past decades. This section summarizes the research efforts with respect to the qualities purposed and trusted traceability.

Section 2.5.1 provides an overview of empirical work that identified and investigated traceability problems in industrial practices with respect to purposed and trusted traceability. The remaining part of this section discusses approaches that have been suggested in order to support purposed and trusted traceability. The approach discussion distinguishes three types of approaches, namely definitional, anticipatory, and analytical approaches. Section 2.5.2 discusses definitional approaches, which are used to remedy the problem that purposed traceability is usually difficult to define within a project spe-

2. Software Traceability

cific context. Existing anticipatory approaches are discussed in Section 2.5.3, which are used to prevent problems with respect to purposed and trusted traceability. Analytical approaches are used to detect these problems and are discussed in Section 2.5.4.

2.5.1. Empirical Work on Traceability Problems

This section summarizes the empirical traceability research efforts that have been made to understand and systematize the challenges to successfully implement purposed traceability in software projects that can be trusted.

In 1994, an extensive study on the so-called traceability problem was conducted [Gotel and C. Finkelstein 1994]. The study involved around one hundred software development practitioners, holding a variety of positions within a large organization, with experience ranging between 0.75 and 30 years on a variety of project types. They identified various reasons for that problem, such as lack of training and guidance in traceability practice, failure to follow standard practices, undefined traceability roles, lack of coordination and cooperation between people responsible for different artifacts, and inadequate information about how people contributed to traceability data.

Arkley and Riddle identified a lack of motivation for requirements traceability and a lack of understanding for how to employ traceability [Arkley and Riddle 2005]. The lack of understanding was mainly the result of communication issues between developers and the quality team, which failed to properly communicate the purpose traceability. The lack of motivation was caused by the fact that developers do not perceive immediate benefits from traceability. Developers considered traceability as a bureaucratic nuisance imposed by the quality team. In [Arkley and Riddle 2006], the authors found that the motivation can be improved by tailoring traceability to stakeholder specific needs.

In [Ramesh et al. 1995], the problem of relatively high costs of creating and maintaining traceability was identified, which can only be compensated by higher quality and reduced overall costs if traceability is applied purposefully. Leading to a traceability cost-benefit problem. Traceability cost drivers such as team size, productivity/experience of the personnel, project complexity, and requirements volatility were identified in [Ingram and Riddle 2012]. Heindl and Biffel found that the trace link creation effort with respect to time depends on the traceable artifacts [Heindl and Biffel 2005]. While creating trace links from requirements to source code methods lasted took on average 45 minutes per requirement, the creation of trace links from requirements to source code classes took on average 10 minutes per requirement. Cleland-Huang et al. assumed an average trace link creation time of 15 minutes for their case studies [Cleland-Huang et al. 2004]. Capturing software development activities to automatically generate traces may reduce trace capturing effort [Marcus and J. Maletic 2003; Hayes et al. 2007; De Lucia et al. 2008; Cleland-Huang et al. 2007; Omoronyia et al. 2011; Delater and Paech 2013]. Though neglecting traceability completely or capturing traces in an unstructured manner to re-

duce costs will lead to reduced system quality, expensive iterations of defect corrections, and increased project costs [Pohl 1996; Dömges and Pohl 1998].

Several studies have investigated the negative impact of inadequate traceability on software development. Researchers found that wrong granularity can lead to over-complex or inadequate traceability graphs, and thereby leads to project over-runs or software failures [Mäder et al. 2009b; Leffingwell 1997]. Serious traceability deficiencies were also found in safety critical software systems [Mäder et al. 2013; Panesar-Walawege et al. 2010]. Both studies reported problems, such as missing or redundant trace links, and problems in trace granularity.

2.5.2. **Definitional Approaches**

Ramesh identified two general groups of traceability users, whom he refers to as low-end and high-end traceability users [Ramesh 1998]. 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 [Ramesh and Jarke 2001]. They pointed out that traceability links should be strongly typed in order to avoid semantic misinterpretations.

2.5.3. **Anticipatory Approaches**

Several authors have conducted empirical research on requirements traceability and argue the need for planned traceability and defined traceability strategies. Gotel and A. Finkelstein argue in [Gotel and A. Finkelstein 1997] that the knowledge about stakeholders that contributed to traced artifacts helps improving traceability.

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 as advocated in [Mäder et al. 2009a]. The traceability information model concept is similar to conceptual traceability model concept, which was proposed in [von Knethen et al. 2002]. Arkley and Riddle conducted a case study in [Arkley and Riddle 2006] 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 (ii) the traceability information model was systematically tailored to the identified needs.

Dömges and Pohl propose a more holistic approach to employ project-specific traceability [Dömges and Pohl 1998]. Rather than reducing traceability to the definition of permitted artifacts and link types, the definition of project-specific trace strategies is advocated. Thereby, trace capture and usage strategies define what data should be captured and used, in which situation, by whom, and how. Additionally, the authors developed the framework PRIME-RT, which provides integrated traceability guidance

2. Software Traceability

by automatically reminding, enforcing, and controlling a project-specific traceability strategy [Pohl et al. 1999]. The authors reported that the application of this framework in prototypical experiments lead to better traceability and higher product quality. Though, PRIME-RT supports the application of traceability strategies, the authors did not discuss what is required to define an adequate traceability strategy, which in turn is necessary to successfully employ the strategy.

2.5.4. Analytical Approaches

Various researchers have proposed traceability metrics to characterize traced software artifacts. For example, Pfleger and Bohner proposed software maintenance metrics for traceability graphs [S. Pfleger and S. Bohner 1990]. They distinguished 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. proposed the use of linkage statistic metrics [Costello and Liu 1995]. Dick extended the idea of analyzing traceability graphs by introducing trace link semantics, which he calls rich traceability. The main advantage of his approach is that propositional reasoning can be applied to analyze traceability relationships for their consistency [Dick 2002]. Hull and Dick carried on with the idea of analyzing rich traceability graphs and proposed further metrics: *breadth* is related to the coverage and measures the progress of a development phase, *depth* measures the number of layers making it a global metric, *growth* is related to the potential change impact, *balance* measures the distribution of growth factors, *latent change* measures the impact on a change [Hull et al. 2011]. While all of the proposed requirements traceability metrics were meant to measure specific characteristics of the requirements traceability graph, little empirical evidence is available on how and to what extent these metrics support practitioners with activities such as requirements planing and requirements impact analysis. Instead of calculating traceability metrics, Canfora and Cerulo employed information retrieval algorithms for the purpose of impact analysis [Canfora and Cerulo 2005]. The authors leveraged the description of requirements changes to automatically analyze the impact of a requirements change. Briand et al. analyzed traceability information between design and test data to automatically characterize design changes [Briand et al. 2002].

For safety-critical software systems, a safety case needs to be compiled in order to argue that the system is safe for use. Traceability can be used to establish evidence for a safety case. Accordingly, traceability is considered an important means to support safety argument [Kelly 1999]. However, a safety argument can only supported through traceability if its suitability can be proven.

There are two approaches that support automatic traceability assessment. In [Ridder-

hof et al. 2007], Ridderhof et al. proposed a methodology to establish a safety argument based on traceability for the automotive domain and its domain specific safety standard ISO 26262 [ISO 26262-6:2011]. For the sake of establishing a safety argument, evidence is established based on traceability. Based on Object Constraint Language (OCL) constraints, the authors formulated rules that check a project’s traceability implementation for missing trace links. Due to the fact, that the authors focus on a specific safety domain only, the proposed verification rules do not provide a comprehensive assessment. Therefore, their approach is not capable of identifying all possible traceability problems. The proposed approach cannot be regarded generalizable, because it can only be applied to design artifacts represented in UML. Panesar-Walawege et al. conceptually modeled the chain of evidence for safety arguments based on the general purpose safety standard IEC 61508 [Panesar-Walawege et al. 2010]. As part of this chain of evidence, the authors derived a Traceability Information Model (TIM) that addresses traceability related requirements of the standard. Similarly to Ridderhof et al., the authors formulated OCL constraints to validate the traceability implementation for missing trace links. The empirical results of the case study showed that the assessed project suffered from missing trace links. The proposed approach also lacks generalizability, because it can only be applied to design artifacts represented in SysML.

The remaining approaches were solely based on manual traceability assessments. Mäder et al. validated the traceability of project data that were submitted the FDA in [Mäder et al. 2013]. Although, the authors neither supported systematic nor reproducible assessments, the empirical results of the study indicate that safety-critical projects in the medical domain suffer from incomplete and incorrect traceability too, which make a safety argument at least more difficult for the manufacturer. Kornecki and Zalewski reported on a case study, where development tools were assessed for qualification purpose. The qualification of development tools is required, if those tools are used for the development of safety-critical software systems. Although, the authors reported that they qualitatively assessed the tool’s traceability, they did not provide a systematic or reproducible assessment procedure. The reported results indicate that “important aspects [...] were not properly captured or were simply lost in the translation” [Kornecki and Zalewski 2005].

The traceability assessment approaches discussed so far, focused on the assessment of the implemented traceability implementation data. The actual traceability implementation process was not explicitly considered. This was done by Regan et al. in [Regan et al. 2014]. A traceability implementation process assessment was proposed for the development of medical devices in accordance with various medical safety standard. Based on these standards, the authors defined traceability best practices such as bidirectional traceability between each change request and relevant problem report. Inspired by the Capability Maturity Model Integration (CMMI) model, Casey and Mc Caffery proposed a traceability process assessment and improvement model *med-trace* for the development

of medical devices. Based on these assessments, an action plan of traceability process improvements can be derived. The authors report the results of two case studies, where the traceability could be improved successfully.

2.6. Critique of the State of the Art

It can be concluded that intensive research effort has been conducted within the area of traceability. Over the past years, a variety of traceability approaches has been proposed to address the goals related to a purposed traceability that can be trusted. However, the discussed traceability approaches suffer from a number of shortcomings. These shortcomings are summarized in the following.

Purposed. Existing anticipatory approaches provide effective means to plan required traceability for a project [F. Pinheiro and Goguen 1996; von Knethen et al. 2002; Mäder et al. 2009a]. However, planning for purposed traceability is not yet supported by any of these approaches. The proposed models do not provide support for linking the required traceability information to its purpose. The output of the existing anticipatory approaches can not be used to specify the target state of a purposed traceability implementation. However, specifying such a target state is a prerequisite to assess the fitness for purpose of a traceability implementation.

Trusted. The most important shortcoming of existing analytical traceability approaches is the lack of systematic and comprehensive guidance on how to determine if a traceability implementation is fit for its purpose. Existing publications of analytical traceability approaches are confined to the presentation of empirical observations, without providing precise and applicable assessment criteria [Kornecki and Zalewski 2005; Ridderhof et al. 2007; Panesar-Walawege et al. 2010; Casey and Mc Caffery 2011; Mäder et al. 2013]. Determining the fitness for purpose of a project's traceability implementation is essential for establishing trust. Although, existing definitional approaches provide precise definitions of traceability semantics, they lack definitions of the fitness for purpose [Ramesh and Jarke 2001; Dick 2002].

Automation. There are two analytical approaches that support automatic assessments. However, both approaches have a very strict limitation. They can only be applied to design artifacts, which are represented as UML [Ridderhof et al. 2007] or SysML models [Panesar-Walawege et al. 2010]. Thus, a holistic traceability assessment of all project artifacts is not supported by these approaches. The remaining existing analytical traceability approaches are based on manual assessments [Kornecki and Zalewski 2005; Casey and Mc Caffery 2011; Mäder et al. 2013; Regan et al. 2014], which are time-consuming and costly for complex traceability graphs. Since traceability is subject of gradual decay,

the fitness for purpose of a project's traceability implementation needs to be assessed in a continuous and timely manner, in order to ensure that it remains trustworthy. The current lack of automated traceability assessments techniques makes it unlikely to completely assess the fitness for purpose of complex traceability implementations in a timely manner.

These shortcomings are emphasized by the fact that current industrial traceability implementations feature serious deficiencies, which often remain undetected. Even safety-critical projects fail to implement traceability that is fit for the safety purpose [Panesar-Walawege et al. 2010; Mäder et al. 2013]. To remedy the identified shortcomings, a novel traceability assessment approach is presented in the following three chapters.

3. The Traceability Assessment Approach

This chapter outlines the proposed traceability assessment approach. Based on the critique of the state of the art (see Section 2.6), challenges for the traceability assessment approach are derived in Section 3.1. Since a profound understanding of the subject is essential to effective assessment, the general characteristics of a software project's traceability implementation are discussed in Section 3.2. Section 3.3 presents an overview of the assessment approach. The two main parts of the approach are introduced. Which challenge is supposed to be addressed by what part of the proposed assessment approach for solving the identified problems within the critique of the state of the art, is also discussed within this chapter.

3.1. Traceability Assessment Challenges

The critical discussion of the state of the art in Section 2.6 has revealed that existing anticipatory traceability approaches do not provide adequate means to plan for purposed traceability. Existing analytical and definitional approaches are not able to assess the fitness for purpose of a project's traceability implementations. Shortcomings were identified with respect to the attributes purposed, trusted, and automation. This section introduces challenges that need to be addressed by a new traceability assessment approach to remedy the identified shortcomings.

3.1.1. Purposed

A traceability target state must be planned before a project's traceability implementation can be assessed. The specification of required traceability information [von Knethen et al. 2002; Mäder et al. 2009a] is commonly used for traceability planning [Mirakhorli and Cleland-Huang 2011; Delater and Paech 2013; Nejati et al. 2012]. As discussed in Section 2.6, existing approaches [von Knethen et al. 2002; Mäder et al. 2009a] do not provide support for linking the required traceability information to its purpose. The lack of support for planning purposed traceability leads to the following challenge:

Challenge 1 (Support the planning for purposed traceability). *The approach shall support the planning for purposed traceability so that all required traceability information*

3. The Traceability Assessment Approach

is justified by one or many purposes.

Traceability is commonly implemented for multiple purposes (e.g., safety analysis, change impact analysis, compliance verification, and coverage analysis). Different purposes mostly require different sets of trace links. Hence, the traceability planning needs to unify the various traceability requirements of these multiple purposes. This leads to another challenge:

Challenge 2 (Support the planning for multiple purposes). *The approach shall support the planning for multiple purposes and unify all relevant traceability requirements.*

To ensure that a traceability target state can be specified, these two challenges need to be addressed by the assessment approach.

3.1.2. Trusted

To establish trust, a project’s traceability implementation needs to be assessed for its fitness for purpose. This requires a common understanding of the term “fitness for purpose”. As discussed in Section 2.6, existing definitional approaches lack a precise definition of the term fitness for purpose [Ramesh and Jarke 2001; Dick 2002]. This leads to the following challenge:

Challenge 3 (Define fitness for purpose). *The assessment approach shall identify relevant quality attributes and provide comprehensible definitions of these attributes with respect to the term fitness for purpose.*

In order to conduct systematic traceability assessments, the definition of clear assessment criteria is required. To ensure reproducible assessment results, assessment criteria need to be defined in a way that does not leave any room for interpretation. As discussed in Section 2.6, existing analytical approaches are lacking clear assessment criteria [Kornecki and Zalewski 2005; Ridderhof et al. 2007; Panesar-Walawege et al. 2010; Casey and Mc Caffery 2011; Mäder et al. 2013]. Therefore, the following challenge needs to be addressed as well:

Challenge 4 (Provide clear assessment criteria). *The assessment approach shall provide clear criteria that can be used to reproducibly assess a project’s traceability implementation for its fitness for purpose.*

An important goal of the assessment is to detect concrete traceability problems. However, detecting only *some* traceability problems is not sufficient. The assessment needs to ensure that *all* traceability problems can be detected. This implies that the assessment criteria need to cover all types of traceability problems. Existing analytical approaches are restricted to a very limited number of traceability problems [Ridderhof et al. 2007; Panesar-Walawege et al. 2010; Mäder et al. 2013]. Accordingly, the following challenge can be derived:

Challenge 5 (Detect all relevant traceability problems). *The assessment approach shall provide a complete set of assessment criteria that cover can detect all relevant traceability problems.*

These three challenges are required to be addressed by the assessment approach to verify the fitness for purpose of a project’s traceability implementations.

3.1.3. Automation

Since traceability is subject to gradual decay, the fitness for purpose of a project’s traceability implementation needs to be assessed in a continuous manner to ensure that it remains trustworthy. Existing approaches for traceability assessment are either manual [Kornecki and Zalewski 2005; Casey and Mc Caffery 2011; Mäder et al. 2013; Regan et al. 2014] or restricted to formalized design artifacts [Ridderhof et al. 2007; Panesar-Walawege et al. 2010]. To ensure continuous traceability assessments of all project artifacts, the traceability assessment needs to be automated and support all project artifacts. This leads to the following challenge:

Challenge 6 (Support automated traceability assessment). *The assessment approach shall support the automated assessment of a project’s entire traceability implementation.*

Addressing these challenges is required in order to support the automated traceability assessments in a continuous manner.

3.2. Characterizing Traceability Implementations

Assessing a project’s traceability implementation effectively requires a profound understanding of its characteristics. These characteristics are discussed in this section. Traceability is implemented through capturing trace links, which establish a directed association between two artifacts (see Definition 3). A variety of traceability characteristics emerge from establishing trace links between artifacts. Figure 3.1 provides a meta-model characterizing these fundamental concepts in Unified Modeling Language (UML) notation. Throughout this thesis, elements of this model are referred to as Traceability Implementation Data (TID). In the following, each **element** of the meta-model will be discussed in detail. Throughout this discussion, functions are introduced to formalize the relationships among the TID elements. These elements and functions will later be used to define generalized assessment rules, which are abstracted from a concrete traceability implementation, and thus, generally applicable to any software project.

Artifact Type. An artifact type represents a particular group of artifacts sharing a common set of characteristics. Let $I_{\mathcal{A}}$ be the set of all implemented artifact types. For

3. The Traceability Assessment Approach

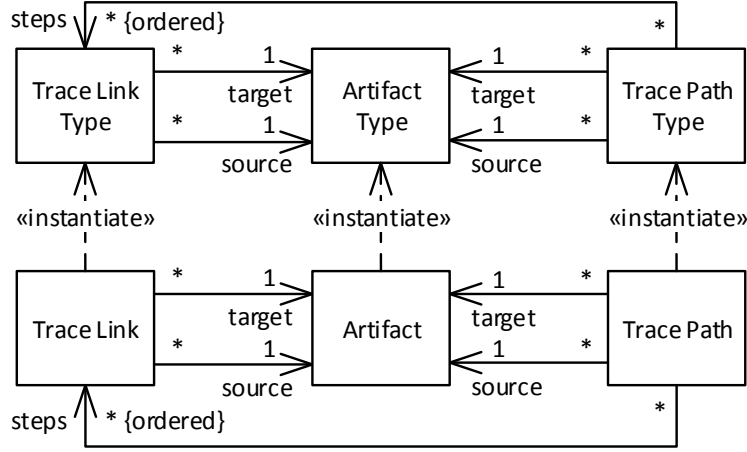


Figure 3.1.: Meta-model of relevant Traceability Implementation Data (TID) characterizing the fundamental concepts of a project's traceability implementation

example, an artifact type software requirement refers to the group of all artifacts that specify a software related requirement for the system to be developed.

Artifact. An artifact refers to any output, whether final or not, that is produced or maintained throughout the entire software lifecycle. Let I_A be the set of all implemented artifacts. The instantiate dependency between the artifact type and the artifact implies that each artifact is an instance of an artifact type and each artifact type is a classifier of an artifact. In the following, this instantiate dependency will be represented by the function $implements : I_A \rightarrow I_{\mathcal{A}}$. This function maps any artifact a_i to the artifact type that is instantiated by a_i so that $implements(a_i) = \{\text{the classifier of the artifact } a_i\}$.

Trace Link Type. A trace link type represents a particular group of trace links sharing two common characteristics. First, the source artifact of each trace link belongs to one specific artifact type, and second, the target artifact of each trace link belongs to one specific artifact type. Accordingly, each trace link type is associated with two artifact types, an originating artifact type holding the role source and a destination artifact type holding the role target. These associations are represented by the following two functions. The function $source : I_{\mathcal{L}} \rightarrow I_{\mathcal{A}}$ maps any trace link type l_i to its source artifact type so that $source(l_i) = \{\text{the source artifact of } l_i\}$. The function $target : I_{\mathcal{L}} \rightarrow I_{\mathcal{A}}$ maps any trace link type l_i to its target artifact type so that $target(l_i) = \{\text{the target artifact of } l_i\}$. Let $I_{\mathcal{L}}$ be the set of all implemented trace link types. In the following, a trace link type is denoted as $I_{\mathcal{A}} \xrightarrow{I_{\mathcal{L}}} I_{\mathcal{A}}$.

Trace Link. A trace link is a directed association, established between two artifacts. Let $I_{\mathcal{L}}$ be the set of all implemented trace links. In this thesis, a trace link is denoted as

3.2. Characterizing Traceability Implementations

$I_A \xrightarrow{I_L} I_A$. A trace link is associated with two artifacts, an originating artifact holding the role source and a destination artifact holding the role target. In the following, these relationships will be represented by two functions. The function $source : I_L \rightarrow I_A$ maps any trace link l_i to its source artifact so that $source(l_i) = \{\text{the source artifact of } l_i\}$. The function $target : I_L \rightarrow I_A$ maps any trace link l_i to its target artifact so that $target(l_i) = \{\text{the target artifact of } l_i\}$. The instantiate dependency between the trace link type and the trace link implies that each trace link is an instance of a trace link type and each trace link type is a classifier of a trace link. In the following, this instantiate dependency will be represented by the function $implements : I_L \rightarrow I_L$. This function maps any trace link l_i to the trace link type that is instantiated by l_i so that $implements(l_i) = \{\text{the classifier of the trace link } l_i\}$.

Trace Path Type. A trace path type results from a sequence of one to many implemented trace link types. Let I_p be the set of all implemented trace path types. Accordingly, a trace path type is denoted as a sequence of trace link types: $I_{\mathcal{A}} \xrightarrow{I_L} I_{\mathcal{A}} \dots I_{\mathcal{A}} \xrightarrow{I_L} I_{\mathcal{A}}$. Thus, a trace path type represents an ordered sequence of trace link types. Similar to a trace link type, the originating artifact type holds the role source and the destination artifact type holds the role target. These associations are represented by the following two functions. The function $source : I_p \rightarrow I_{\mathcal{A}}$ maps any trace path type p_i to its source artifact type so that $source(p_i) = \{\text{the source artifact type of } p_i\}$. The function $target : I_p \rightarrow I_{\mathcal{A}}$ maps any trace path type p_i to its target artifact type so that $target(p_i) = \{\text{the target artifact type of } p_i\}$.

Trace Path. The trace path element refers to a sequence of one to many implemented trace links. Let I_P be the set of all trace paths in a software development project, which result from all implemented trace links. Accordingly, a trace path is denoted as $I_A \xrightarrow{I_L} I_A \dots I_A \xrightarrow{I_L} I_A$. Similar to a trace link, the originating artifact of a trace path holds the role source and the destination artifact holds the role target. The function $source : I_P \rightarrow I_A$ maps any trace path p_i to its source artifact so that $source(p_i) = \{\text{the source artifact of } p_i\}$. The function $target : I_P \rightarrow I_A$ maps any trace path p_i to its target artifact so that $target(p_i) = \{\text{the target artifact of } p_i\}$. The instantiate dependency between the trace path type and the trace path implies that each trace path is an instance of a trace path type and each trace path type is a classifier of a trace path. In the following, this instantiate dependency is represented by the function $implements : I_P \rightarrow I_P$. This function maps any trace path p_i to the trace path type that is instantiated by p_i so that $implements(p_i) = \{\text{the classifier of the trace path } p_i\}$.

3.3. Overview of the Assessment Approach

Assessing a project’s traceability information for its fitness for purpose is closely related to the traceability qualities purposed and trusted (see Section 2.4). It verifies that the implemented traceability is suitable with respect to the project specific traceability goals. The verified traceability implementation can be trusted to support the project specific traceability goals.

In practice, there are several drivers for implementing traceability [Mäder et al. 2009b]. General projects implement traceability to support software engineering activities that require traceability. In this case, the provided value is the main driver for implementing traceability. In safety-critical domains (i.e., avionic, public transportation, or medical devices), the creation of traceability is required by regulatory authorities. Accordingly, the traceability drivers *regulation* and *value* are distinguished in this thesis. Depending on a project’s driver for implementing traceability, its fitness for purpose can have different implications. If traceability is required by a regulatory authority to demonstrate the safety of the developed product, its fitness for purpose is inevitable to get approval for the release to market. Otherwise, it determines whether or not the associated value of a traceability goal (i.e., the support of a specific software engineering activity) can be provided by the implemented traceability.

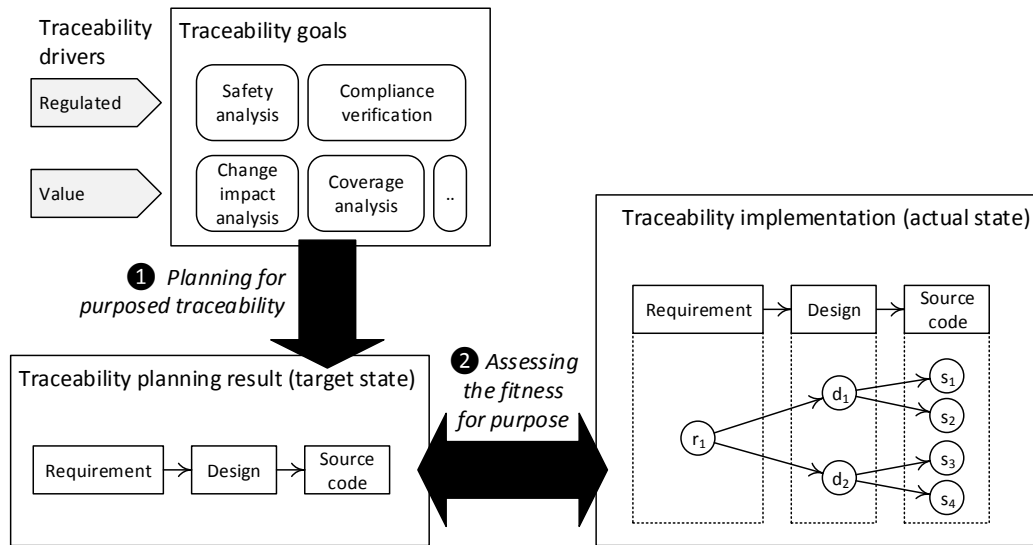


Figure 3.2.: Overview of the traceability assessment approach

Figure 3.2 provides an overview of the proposed traceability assessment approach. The approach consists of two parts: **1** *planning for purposed traceability* and **2** *assessing the fitness for purpose*. Part **1** is a prerequisite of the assessment approach and specifies

the target state of a traceability implementation. Without specifying a target state, a reference for assessing the actual state of a project's traceability implementation would be missing. The target state of a traceability implementation is derived from project specific traceability goals through the planning of purposed traceability. As discussed above, these traceability goals are driven by regulation or value. A detailed discussion of the preliminary first part of the assessment approach is provided in Chapter 4.

Part ② represents the core part of the assessment approach. It systematically assesses the actual state of a project's traceability implementation with respect to the preliminarily derived target state to determine its fitness for purpose. For this purpose an analytical Traceability Assessment Model (TAM) is presented. The abstract term fitness for purpose is decomposed into explicit quality attributes, which are respectively defined. The TAM provides definitions on how these qualities can be assessed in project specific TID. As an integral part of the TAM, a comprehensive classification of atomic traceability problems is provided with respect to the identified quality attributes. Chapter 5 presents the assessment approach in detail. The implementation of prototype for conducting automatic traceability assessments is provided in Chapter 6.

3.4. Usage Scenarios of Traceability Assessment

Planning for purposed traceability and assessing the fitness for purpose of traceability implementations can be useful for different reasons. This section discusses six major usage scenarios of traceability assessment. Four scenarios (scenario 1-4) are primarily relevant for the traceability driver *regulation*. The two remaining scenarios (scenario 5-6) are primarily relevant for the traceability driver *value*.

Scenario 1: compliance to relevant guidelines. Safety critical software products need to be certified by the responsible authority before they can be released to the market. During this certification process, the authority checks whether or not the developed systems can be considered as safe to be used. As part of this certification process, certifiers check compliance of the implemented traceability with traceability requirements of the relevant guideline(s). The introduced assessment approach can be used to evaluate the projects conformance to the relevant guidelines with respect to traceability.

Scenario 2: continuous certification. Consistently maintaining a project in a ready-to-certify state is challenging, but certainly not impossible [Farail et al. 2006]. It requires a rigorous verification process built into the development environment, continuous integration, and accurately maintained traceability, available at any time to support the certification process. The proposed approach supports ongoing analysis of the traceability fitness for purpose of a system with respect to the relevant guidelines.

3. *The Traceability Assessment Approach*

Scenario 3: migration to a new or revised guideline. When an existing product is introduced into a new market, it may be necessary to certify the product under a new guideline. Similarly, existing guidelines may be revised (for example, DO-178B → DO-178C) and the new version will immediately become relevant for product development. In such scenarios the existing traceability model is updated to reflect the new and/or modified guidelines, and a gap analysis is performed between the updated and original traceability model. The approach provides support for planning the necessary traceability implementation changes due to the new or revised guideline. Additionally, the approach provides support for identifying traceability problems introduced by the adoption of a new or revised guideline.

Scenario 4: conformance to multiple guidelines. Products are often released into multiple markets governed by different guidelines. Similarly, a single product may contain components governed by different guidelines. In both cases, the product needs to comply to multiple guidelines. This introduces the need for creating a merged traceability model for two or more guidelines. To find the high watermark, i.e., the minimum set of traceability requirements that, if followed, will satisfy all relevant guidelines, traceability requirements need to be merged and contradictions need to be addressed [Gordon and Breaux 2013]. This approach derives a single set of required traceability information from different traceability goals. It supports the systematic planning for multiple traceability purposes. Thus, the approach provides support for identifying traceability problems across multiple guidelines.

Scenario 5: feasibility of software lifecycle activities. Traceability is required by numerous software lifecycle activities such as, for example, change impact analysis, coverage analysis, and compliance verification. The set of trace links required by one activity can be different to the set required by another activity. Thus, the feasibility of a particular activity depends on the completeness of the respective set of trace links. The approach can be used to determine the completeness of these sets of trace links. Thus, the approach provides support for determining the feasibility of software lifecycle activities. The approach can also be used to plan for traceability that ensures the feasibility of software lifecycle activities.

Scenario 6: cost effective traceability implementation. The manual creation of trace links is cost-intensive. Creating trace links takes on average 15 minutes [Cleland-Huang et al. 2004]. Depending on the artifact types, the average creation times can vary between 10 and 45 minutes [Heindl and Biffel 2005]. Hence, the creation of superfluous trace links should be avoided in order to ensure a cost effective traceability implementation. The planning for purposed traceability provides explicit recommendations for cost effective traceability implementation. Further, the approach can be used to identify and

3.4. Usage Scenarios of Traceability Assessment

eliminate superfluous trace links within existing traceability implementations to reduce the traceability maintenance costs.

These six scenarios describe the context for the assessment approach presented in the following three chapters. Additionally, these scenarios are used as a reference to derive research questions for the evaluation of the assessment approach.

4. Planning for Purposed Traceability

Planning for purposed traceability is a prerequisite for assessing the fitness for purpose of a project's traceability assessment. The result of the planning specifies the target state of a traceability implementation, which is used as assessment reference. This chapter presents a systematic approach for planning purposed traceability. As elaborated in Section 3.3, traceability drivers can be regulation or value. The traceability planning approach provides the capability to address both traceability drivers. Traceability driver specific planning considerations are explicitly highlighted.

The planning approach consists of two activities namely *identifying traceability requirements* and *specifying traceable artifact types*. The first activity derives traceability requirements from project specific traceability goals (see Section 4.1). The second activity specifies what artifact types should be traceable and how traceability should be established between these artifact types (see Section 4.2). Section 4.3 discusses how the produced planning output can be used to justify the purpose of each trace link type. Section 4.4 summarizes the planning approach with respect to the challenges that are related to purposed traceability (see Section 3.1.1).

4.1. Identifying Traceability Requirements

The main objective of the traceability planning approach is to specify purposed traceability. Therefore, the traceability planning needs to be driven by a project's intended traceability usage, which represents the project specific purpose. To ensure that the traceability planning is driven by the intended traceability usage, a goal oriented approach is proposed to identify traceability requirements. The usage of goals is a common technique in software engineering for identifying and justifying software requirements. Therefore, the concept of goals is also used for identifying traceability requirements. In a broad context of software engineering, goals are considered as high-level objectives of the business, organization, or system, which capture why the development of a software system is necessary [Anton 1996]. For the traceability planning approach, the concept of goals is used in a much more restricted way. Software traceability aims to support other software engineering activities throughout the software lifecycle. Therefore, within the context of software traceability, the concept of goals is restricted to objectives related to a software lifecycle activity.

To ensure that traceability requirements are produced in a systematic fashion, a

4. Planning for Purposed Traceability

precise definition of the expected output is required. Hence, a model for traceability requirements is defined in Section 4.1.1. The remainder of this section provides a detailed discussion of the required steps to systematically identify traceability requirements. The activity “identifying traceability requirements” consists of the following steps:

1. *Identifying software lifecycle related goals.* Software traceability aims at supporting software engineering activities throughout the software lifecycle. Thus, any software lifecycle related stakeholder objective is potentially relevant for the elicitation of traceability requirements. These objectives are identified in a goal-oriented manner (see Section 4.1.2).
2. *Identifying goal specific activities.* Software engineering activities are performed by agents throughout the software lifecycle in order to achieve the stakeholders’ goals. To support the analysis for determining if an activity requires traceability, all goal specific activities need to be identified and documented (see Section 4.1.3).
3. *Identifying goals that require traceability.* Only those software lifecycle related goals are relevant for the definition of traceability requirements that require traceability. Thus, the subset of goals is identified that require software traceability to be achievable (see Section 4.1.4).
4. *Identifying traceability implementation activities.* Goals that require traceability can only be achieved if the required traceability is implemented. Hence, this last step identifies traceability implementation activities that need to be performed to establish the required traceability (see Section 4.1.5).

4.1.1. A Model for Traceability Requirements

To ensure that the planning output can be used as a reference for assessments, clear traceability requirements are necessary. The creation of a Traceability Requirements Model (TRM) is advocated. To provide a precise definition of the created traceability requirements, a meta-model for the TRM is defined. Figure 4.1 provides an overview of the defined meta-model as UML class diagram. While the top part of Figure 4.1 depicts general concepts related to the software lifecycle, the bottom part depicts traceability specific concepts. The remainder of the section discusses each meta-model **element** in detail. Throughout the discussion, functions are introduced to formalize the relationships among the TRM elements. These elements and functions will later be used to demonstrate how the purpose of each specified trace link type can be justified.

Goal. The goal element describes stakeholders’ interests which pertain to the software system’s development. Let \mathcal{G} be the set of all goals. Collecting software lifecycle related goals during the traceability requirements planning ensure that it is driven by the stakeholders’ interests rather than the gut feeling of the project manager.

4. Planning for Purposed Traceability

Traceability Goal. A traceability goal represents a subset of the goal element that contains traceability related goals only. For planning the intended use of traceability, the identification of traceability related goals is necessary. Let \mathcal{T}_G be the set of all traceability goals. Since traceability goals are a subset of goals, the mapping of the function *concerns* includes a stakeholder's traceability goals.

Traceability Usage Activity. The element traceability usage activity is a subset of the activity element that represents activities related with using traceability only. Identifying traceability usage activities is important to derive purposed traceability requirements. Let \mathcal{T}_{uc} be the set of all traceability usage activities. Traceability usage activities are performed to achieve traceability goals. This relationship is represented by the function *achieves* : $\mathcal{T}_{uc} \rightarrow 2^{\mathcal{T}_G}$ that maps any traceability usage activity t_i^{uc} to a set of traceability goals that are supposed to be achieved by t_i^{uc} so that *achieves*(t_i^{uc}) = {the traceability goals to be achieved by t_i^{uc} }.

Traceability User. The element traceability user is a subset of the agent element. It represents agents that perform traceability usage activities. Let $\mathcal{T}_{u\mathcal{N}}$ be the set of all traceability users. The relationship that traceability users perform traceability usage activities is represented by the function *performs* : $\mathcal{T}_{u\mathcal{N}} \rightarrow 2^{\mathcal{T}_{uc}}$ that maps any traceability user t_i^{un} to a set of activities that are performed by t_i^{un} so that *performs*(t_i^{un}) = {the traceability usage activities performed by t_i^{un} }.

Traceability Implementation Activity. The element traceability implementation activity is a subset of the activity element that represents activities related with implementing traceability only. Let \mathcal{T}_{IC} be the set of all traceability implementation activities. As discussed in Section 2.3, performing traceability implementation activities enables the performance of traceability usage activities. This relationship is represented by the function *enables* : $\mathcal{T}_{IC} \rightarrow 2^{\mathcal{T}_{uc}}$ that maps any traceability implementation activity t_i^{ic} to a set of traceability usage activities that are enabled by t_i^{ic} so that *enables*(t_i^{ic}) = {the traceability usage activities that are enabled by t_i^{ic} }.

Traceability Implementer. The element traceability implementer is a subset of the agent element. It represents agents that perform traceability implementation activities. Let $\mathcal{T}_{I\mathcal{N}}$ be the set of all traceability implementers. The relationship that traceability implementers perform traceability implementation activities is represented by the function *performs* : $\mathcal{T}_{I\mathcal{N}} \rightarrow 2^{\mathcal{T}_{IC}}$ that maps any traceability implementer t_i^{in} to a set of activities that are performed by t_i^{in} so that *performs*(t_i^{in}) = {the traceability implementation activities performed by t_i^{in} }.

4.1.2. Identifying Software Lifecycle Related Goals

Software traceability aims to support software lifecycle related goals. To understand which goal requires traceability, all goals need to be identified initially. Depending on the traceability driver, different sources of information are relevant.

For safety-critical projects that are driven by regulation, explicit guidelines are provided by the responsible authorities. These guidelines follow a similar structure which starts with a description of the software lifecycle. This lifecycle consists of multiple processes, each composed of activities, their prerequisites, and the produced artifacts. Additionally, a guideline defines the objectives to be fulfilled for demonstrating safety. These objective definitions can be used to directly derive software lifecycle related goals. Stakeholder is the regulatory authority. The authority is concerned with the achievement of these goals to ensure product safety.

For projects where implementing traceability is driven by value, software lifecycle related goals need to be elicited directly. Therefore, relevant stakeholders of the project that may be concerned with the result of any software engineering activity need to be identified. These stakeholders need to be interviewed in order to identify and document their goals. Conducting these interviews requires careful preparation (for example, interview type, questionnaire, briefing, and recording). A detailed interview study that illustrates the application of interviewing techniques to identify stakeholder specific goals, will be discussed in Section 7.3. For this study, stakeholders from 17 software companies were interviewed to identify software lifecycle related goals. However, contributing to the theory of interviewing techniques is out of the scope of this thesis. The existing work on that topic provides comprehensive guidance (see, for example, [Gorden 1980; Bellamy et al. 2006; Fitzpatrick et al. 2009]).

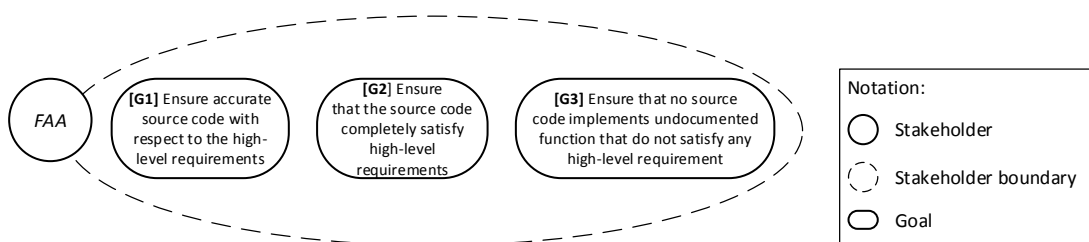


Figure 4.2.: Illustrating autopilot project: goals of the stakeholder Federal Aviation Administration (FAA) that were derived from the guideline DO-178B

To document the planning results, the Goal-oriented Requirement Language (GRL) is leveraged [Amyot 2003]. The GRL provides effective means to document the identified goals and stakeholders in accordance with the TRM meta-model (see Section 4.1.1). For illustration purposes, a fictional software project is used to illustrate the concrete application of each step of the proposed planning activities. In the following, this

4. Planning for Purposed Traceability

fictional project is referred to as *autopilot*, developing an automatic flight control system. Autopilot is safety-critical and regulated by the Federal Aviation Administration (FAA), which applies the guideline [DO-178B] to determine if it will perform reliably in an airborne environment. This means that the guideline DO-178B can be used to derive goals of the stakeholder FAA. Figure 4.2 depicts three goals, **G1**, **G2**, and **G3**, that were derived from the guideline DO-178B. It should be noted that the three goals represent an illustrating excerpt only, and by far do not cover the DO-178B guideline completely.

4.1.3. Identifying Goal Specific Activities

As specified within the TRM meta-model, one or many activities are performed by an agent in order to achieve a goal. These activities need to be identified and documented. Each activity can be analyzed in later steps to determine if it requires traceability.

Similar to the goal identification step, different sources of information are relevant for the identification of goal specific activities. Guidelines explicitly specify activities for regulated projects that need to be performed with respect to the goals. For projects that implement traceability due to the excepted value, two sources of information are relevant. First, software engineers who perform activities throughout the software lifecycle need to be interviewed. These interviews provide insights to the activities that are performed by human agents. Second, the documentation of automated agents needs to be analyzed to understand automatically executed activities.

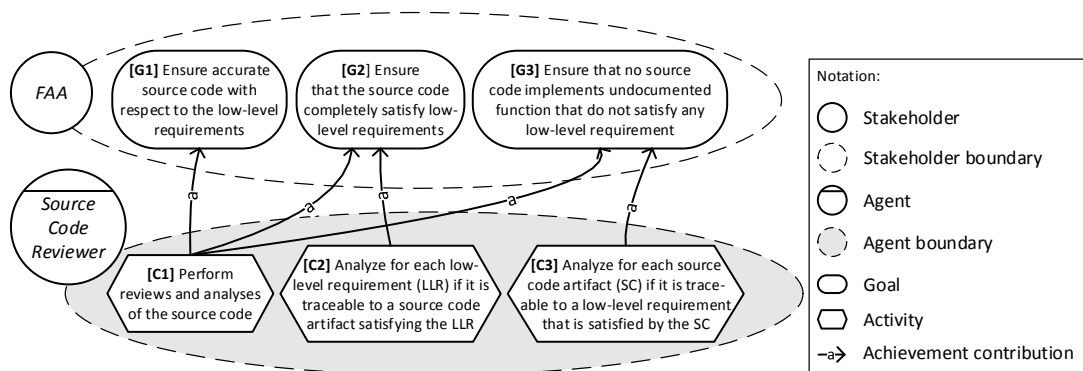


Figure 4.3.: The autopilot project: activities derived from the guideline DO-178B that are supposed to address the goals of the stakeholder Federal Aviation Administration (FAA)

Figure 4.3 depicts the three activities, **C1**, **C2**, and **C3**, that were derived from the guideline DO-178B for the autopilot example. These activities are performed by the source code reviewer. The goal **G1** is achieved by the performance of **C1**, the goal **G2** is achieved by the performance of **C1** and **C2**, and the goal **G3** is achieved by the performance of **C1** and **C3**.

4.1.4. Identifying Goals that Require Traceability

The two preceding steps have identified all the activities that are performed by agents throughout the software lifecycle to achieve the stakeholders' goals. For the identification of traceability requirements, only the subset of traceability goals is relevant. As defined in the TRM meta-model, traceability goals are goals that require traceability. Since the identified activities are performed to achieve the goals, each activity needs to be analyzed to determine if its performance requires traceability. As experienced in a prior study [Rempel et al. 2014], activities that require traceability can be identified systematically by searching for keywords that either refer to traceability in general (i.e., trace, trace link, traceable, or traceability) or that refer to common trace link semantics (i.e., evolve, satisfy, depend on, or verify). Ramesh and Jarke provide a comprehensive classification of possible trace link semantics in [Ramesh and Jarke 2001] that can be used to select appropriate keywords for trace link semantics. Another prior study showed that these keywords can also be used very effectively in interviews with agents and stakeholders to identify activities that require traceability in projects that are driven by the value of implemented traceability [Rempel et al. 2013].

This means that the keyword based search in activity definitions is used to identify those activities that require traceability. As defined in the TRM meta-model, these activities are denoted as traceability usage activities. Since each activity is associated to the goals that are achieved by its performance, the set of traceability goals can directly be derived by searching for all goals that are associated to a traceability usage activity.

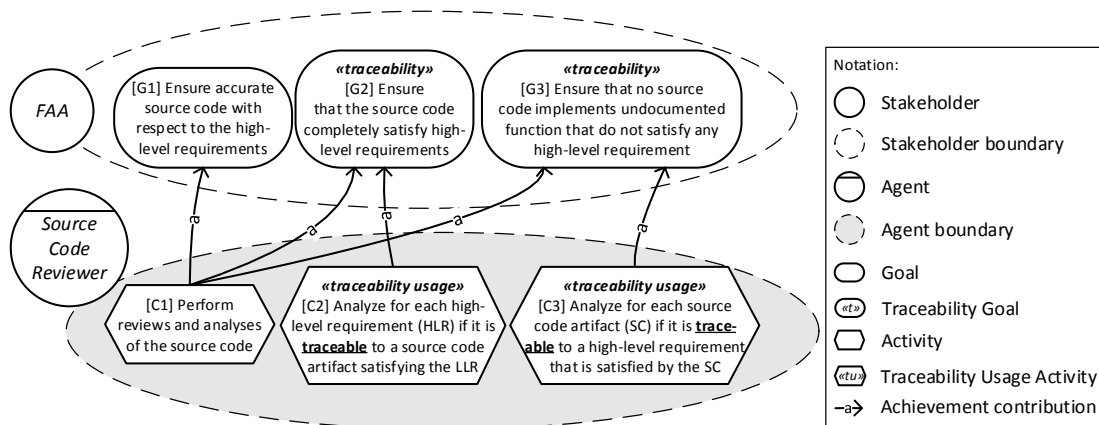


Figure 4.4.: The autopilot project: identified activities that require traceability (traceability usage activities) and their related goals (traceability goals)

Figure 4.4 shows the result of the keyword based search for traceability related goals. The activities C2 and C3 contain the keyword *traceable*. Accordingly, both activities are explicitly marked as traceability usage tasks by the stereotype «traceability usage». The goal G2 is achieved, among others, by the performance of C2 and the goal G3 is achieved,

4. Planning for Purposed Traceability

amongst other, by the performance of C3. Accordingly, the two goals, G2 and G3, are marked as traceability goals by the stereotype «*traceability*».

4.1.5. Identifying Traceability Implementation Activities

Traceability goals can only be achieved if the associated traceability usage activities can be performed. The performance of traceability usage activities requires implemented traceability. As defined in the TRM meta-model, traceability usage activities are enabled by traceability implementation activities. Hence, to finalize the identification of traceability requirements, required traceability implementation activities need to be identified and specified.

Traceability usage activities contain the necessary information for identifying traceability implementation activities. Two kinds of information need to be identified. First, the source artifact type that needs to be traceable. Second, the target artifact type to which traceability needs to be provided. A traceability usage activity can potentially require traceability for multiple pairs of source and target artifact types. In this case, a multiple traceability implementation activities are defined. This approach ensures that each defined traceability implementation activity is atomic, defining one pair that contains a source artifact type and a target artifact type.

As discussed in Section 2.3, a traceability implementation activity consists of the two sub-activities traceability creation and traceability maintenance. This needs to be considered when traceability implementation activities are defined. Traceability can only be created if both, the source and the target artifact, exist. That means, the agent who creates the artifact that is created second is responsible for creating the traceability between the two artifacts. As at the point in the time when the second artifact is created, both artifacts, source and target, are available for creating traceability. Traceability maintenance is required if either the source or the target artifact is maintained. The agent who maintains either the source or the target artifact is responsible to maintain its traceability. This means that three traceability implementation activities need to be specified for each triplet, one creation activity and two maintenance activities. To standardize the definition of traceability implementation activities, the usage of a specification template is advocated.

In Grammar 4.1, a specification template is specified in Backus Naur Form. The template consists of the following parts:

- *Identifier* $\langle id \rangle$. It specifies a unique identifier for the activity.
- *Implementation activity* $\langle impl-act \rangle$. The implementation activity can either be *create* or *maintain*. For the activity *create*, it needs to be specified if the required traceability relation is unidirectional from source to target ($->$), unidirectional from target to source ($<-$) or bidirectional ($<->$). For the activity *maintain*, it needs

$$\begin{aligned}
\langle \text{template} \rangle & ::= \langle \text{id} \rangle \langle \text{impl-act} \rangle \text{'='} \langle \text{source} \rangle \text{'='} \langle \text{target} \rangle \text{'.'} \\
\langle \text{id} \rangle & ::= \text{'['} \langle \text{literal} \rangle \text{']'} \\
\langle \text{impl-act} \rangle & ::= \text{'create' '=' '->' | 'create' '=' '<-'} | \text{'create' '=' '<->' | 'maintain' '=' 'source' | 'maintain' '=' 'target'} \\
\langle \text{source} \rangle & ::= \text{'source' '='} \langle \text{literal} \rangle \\
\langle \text{target} \rangle & ::= \text{'target' '='} \langle \text{literal} \rangle \\
\langle \text{literal} \rangle & ::= \langle \text{char} \rangle \{ \langle \text{char} \rangle \}
\end{aligned}$$

Grammar 4.1: Specification template for traceability implementation activities

to be specified if a change to the source or to the target artifact type triggers the maintenance.

- *Source* ($\langle \text{source} \rangle$). It specifies the source artifact type of the required traceability relation.
- *Target* ($\langle \text{target} \rangle$). It specifies the target artifact type of the required traceability relation.

To demonstrate the practical usage of the specification template for traceability implementation activities, the advocated template is used for the autopilot project. Figure 4.5 shows the defined traceability implementation activities C4, C5, and C6 that enable the traceability usage activities C2 and C3. C4 specifies the creation activity, C5 specifies the maintenance activity for the case that the target artifact is changed, C6 specifies the maintenance activity for the case that the source artifact is changed. The traceability usage activities C2 and C3 are enabled by the same set of traceability implementation activities, as both use the same traceability relation in different directions. Accordingly, the traceability implementation activity C4 specifies the creation of bidirectional traceability.

4.2. Identifying Required Traceability Information

The results of the traceability planning are supposed to define the traceability information that is required to be implemented. The assessment approach uses the required traceability information as a target state. To ensure that the traceability planning output can be used as a target state for assessments, a precise definition of the expected output is required.

4. Planning for Purposed Traceability

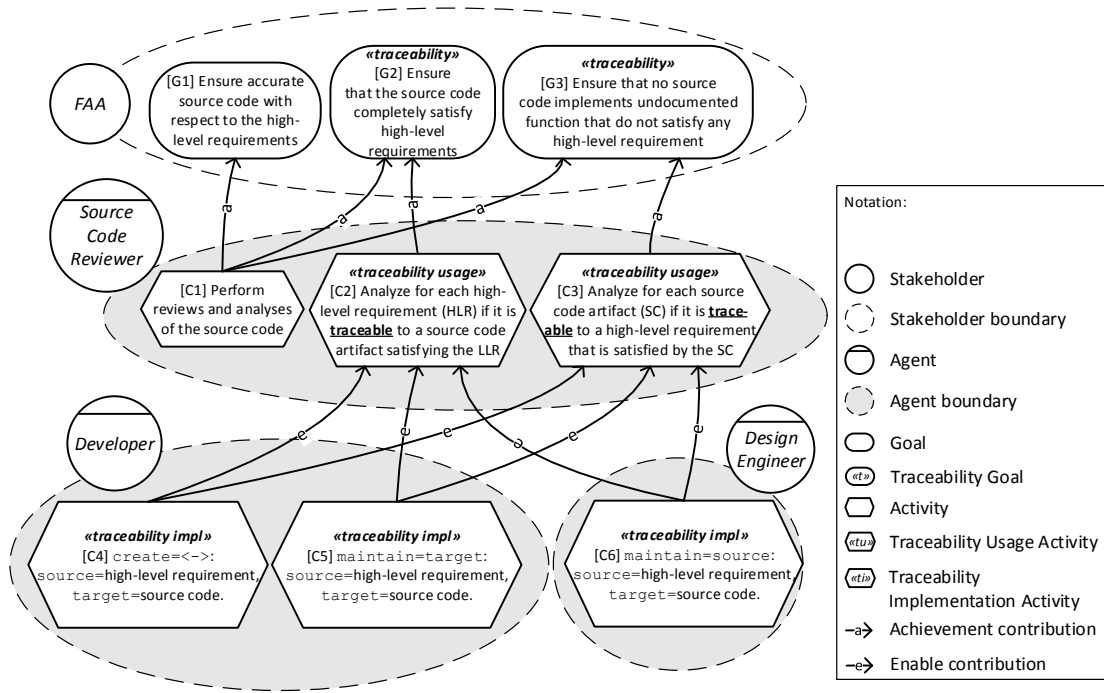


Figure 4.5.: The autopilot project: derived traceability requirements (traceability implementation activities)

To satisfy this need for a precise definition, a model for required traceability information is defined in Section 4.2.1. The remainder of this section provides a detailed discussion of the required steps to systematically specify required traceability information at the required level of granularity. The activity “specifying traceable artifact types” consists of the following steps:

- *Identifying required trace path types.* Required trace path types are derived from the traceability requirements defined within the TRM (see Section 4.2.2).
- *Identifying required trace link types.* Required trace path types consist of sequences of one to many required trace link types. The required trace link types are identified to break down the planning to single trace links (see Section 4.2.3).

4.2.1. A Model for Required Traceability Information

As outlined in Section 2.5.3, the creation of a so called Traceability Information Model (TIM) is the current state of practice for the specification of a project’s traceability implementation target state [Mäder et al. 2009a]. The TIM approach provides for the concept’s required artifact type and for the required trace link type. However, one important concept is missing. A specific traceability purpose may lead to a traceability requirement that demands for a sequence of more than one trace link types. For example,

the verification of requirements is typically ensured by a chain of artifacts. A test case is defined to verify a requirement. This test case is typically executed by a test procedure. Finally, the execution of the procedure leads to a test result. To analyze whether or not the verification of a requirement leads to a positive result, the complete path of trace links needs to be traversed. Another example would be the analysis if all requirements are implemented. Requirements are typically satisfied by design artifacts and these design artifacts are implemented by source code artifacts. To analyze whether or not a requirement is implemented by a source code artifact, the complete path from requirements through the design to the source code needs to be considered for the analysis.

To address this problem, the existing TIM approach is extended by the concept required trace path type. To provide a precise definition of possible instances, a TIM meta-model is provided. Figure 4.6 shows the meta-model as UML class diagram. The remainder of the section discusses each meta-model **element** in detail. Throughout the discussion, functions are introduced formalize the relationships among the TIM elements. These elements and functions will later be used for the definition of generalized traceability assessment rules.

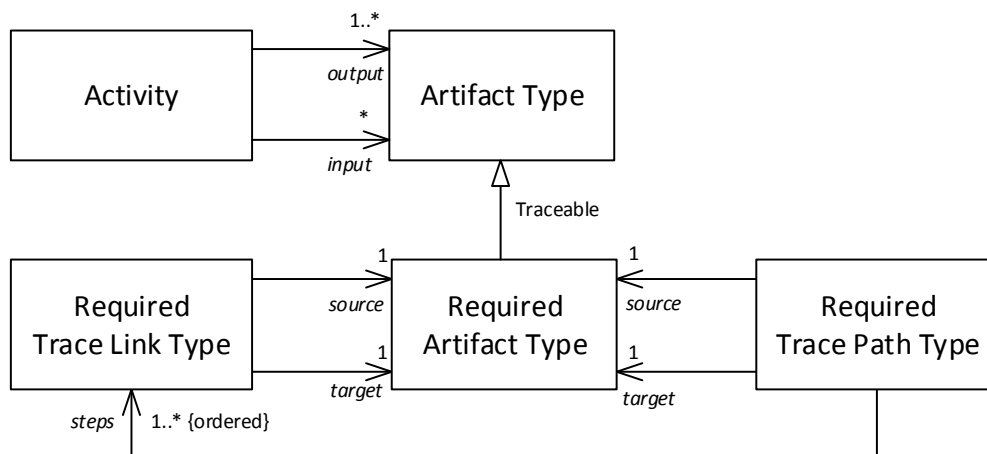


Figure 4.6.: Meta-model of the extended Traceability Information Model (TIM)

Artifact Type. The term artifact refers to any work product that is created throughout the software lifecycle. Accordingly, an artifact type refers to the types of work products.

Activity. Throughout the software lifecycle, a variety of software engineering activities are performed to develop the product. Each activity is characterized by the set of artifact types that it creates. Additionally, an activity may use one or more previously created artifact types. The function $input : C \rightarrow 2^A$ maps any activity a_i to its input artifact types so that $input(a_i) = \{\text{the input artifact types of } a_i\}$. The function $output :$

4. Planning for Purposed Traceability

$C \rightarrow 2^{\mathcal{A}}$ maps any activity a_i to its set of output artifact types so that $output(a_i) = \{\text{the output artifact types of } a_i\}$.

Required Artifact Type. The element required artifact type represents a subset of those artifact types that are required to be traceable. Let $\mathcal{R}_{\mathcal{A}}$ be the set of all artifact types that are required to be traceable.

Required Trace Link Type. A required trace link type represents the types of trace links that need to be created to satisfy a traceability requirement as specified in the TRM. Let $\mathcal{R}_{\mathcal{L}}$ be the set of all required trace link types. A required trace link type refers to a required source artifact type and to a required target artifact type. These associations are represented by the following two functions. The function $source : \mathcal{R}_{\mathcal{L}} \rightarrow \mathcal{R}_{\mathcal{A}}$ maps any required trace link type l_i to its required source artifact type so that $source(l_i) = \{\text{the required source artifact type of } l_i\}$. The function $target : \mathcal{R}_{\mathcal{L}} \rightarrow \mathcal{R}_{\mathcal{A}}$ maps any trace link type l_i to its required target artifact type so that $target(l_i) = \{\text{the required target artifact type of } l_i\}$.

Required Trace Path Type. A required trace path type represents a sequence of trace link types that need to be created in order to satisfy a traceability requirement as specified in the TRM. Let $\mathcal{R}_{\mathcal{P}}$ be the set of all required trace path types. Similar to a required trace link type, the originating required artifact type holds the role source and the destination artifact type holds the role target. These associations are represented by the following two functions. The function $source : \mathcal{R}_{\mathcal{P}} \rightarrow \mathcal{R}_{\mathcal{A}}$ maps any trace path type p_i to its source artifact type so that $source(p_i) = \{\text{the required source artifact type of } p_i\}$. The function $target : \mathcal{R}_{\mathcal{P}} \rightarrow \mathcal{R}_{\mathcal{A}}$ maps any trace path type p_i to its target artifact type so that $target(p_i) = \{\text{the required target artifact type of } p_i\}$.

4.2.2. Identifying Required Trace Path Types

As a first step, required trace path types are identified, as trace path types are the most coarse-grain concepts of the TIM. As discussed in Section 4.2.1, a required trace path type consists of a sequence of one to many required trace link types and the respective required artifact types. Furthermore, traceability implementation activities of the TRM are defined at this level of granularity. The required trace path types can be derived from traceability implementation activities of the TRM that refer to creation activities. Due to the usage of a formalized specification template, the following general derivation rules can be defined:

1. The $\langle literal \rangle$ of the element $\langle source \rangle$ represents the source of the required trace path type.
2. The $\langle literal \rangle$ of the element $\langle target \rangle$ represents the target of the required trace

path type.

3. The $\langle impl-act \rangle$ indicated the direction of the required trace path type. If bidirectional traceability is required, two required trace path types are derived, one from source to target and one from target to source.

These rules can be used to derive a first draft of the TIM, containing required trace path types. Figure 4.7 depicts the application of these derivation rules for the illustrating autopilot project. The left part of the figure shows the traceability implementation activity C4 of the autopilot’s TRM. The right part shows the first draft of the autopilot’s TIM containing two required artifact types HLR and SC as well as two required trace path types HLR ->> SC and SC ->> HLR. The following four derivation rules were applied:

- 1 The required artifact type HLR was derived from the $\langle source \rangle$ element of C4.
- 2 The required artifact type SC was derived from the $\langle target \rangle$ element of C4.
- 3 The required trace path type HLR ->> SC was derived from the $\langle impl-act \rangle$ element of C4 that requires bidirectional traceability.
- 4 The required trace path type SC ->> HLR was derived from the $\langle impl-act \rangle$ element of C4 that requires bidirectional traceability.

The illustrated procedure for deriving required trace path types is repeated accordingly, if a TRM contains multiple traceability implementation activities for creation. The result of this first step is the draft of a TIM, containing required trace path types.

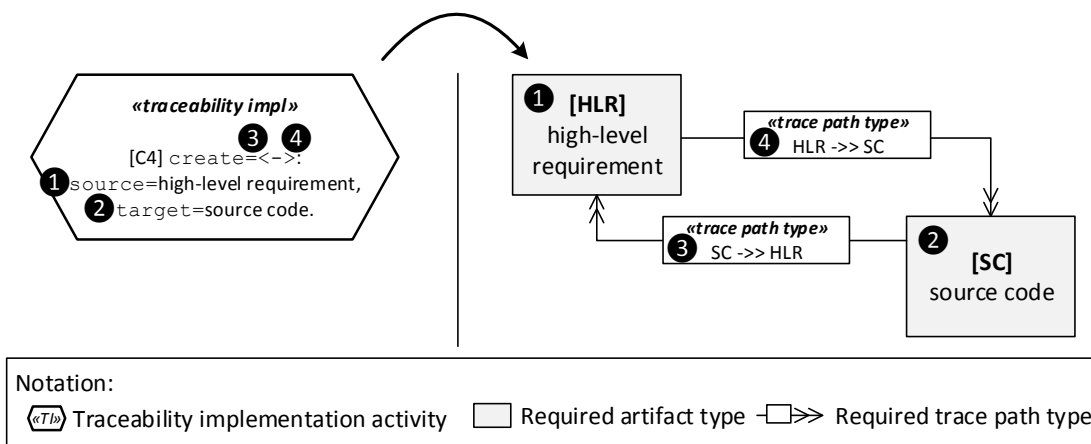


Figure 4.7.: The autopilot project: deriving required trace path types from a traceability implementation activity

4.2.3. Identifying Required Trace Link Types

The previous step derived a draft of the TIM, containing required trace path types. However, this TIM draft does not yet provide all of the required information. It lacks the information, which trace link types are required for creating the required trace path types. As discussed in Section 4.2.1, a required trace path type can consist of a sequence of many required trace link types.

Hence, this section discusses how these required trace link types can be derived for each required trace path type. Initially, a required trace path type provides three kinds of information, the source artifact type, the target artifact type, and the requirement that a path between source and target is required. Starting from the source artifact type, possible trace link types, that originate from the source artifact type, need to be identified. For this purpose, the activities of the software lifecycle are analyzed. As defined in Section 4.2.1, each activity creates one or many artifact types. Additionally, an activity may use one or more previously created artifact types. As the performance of an activity represents an evolution of the software lifecycle, each combination of activity input and output artifact type represents a possible trace link type. To identify the required trace link types for a trace path type, the sequence of activities, that evolve from the source of a trace path type to the target of a trace path type, needs to be identified. This sequence of activities represents the required trace link types of a trace path.

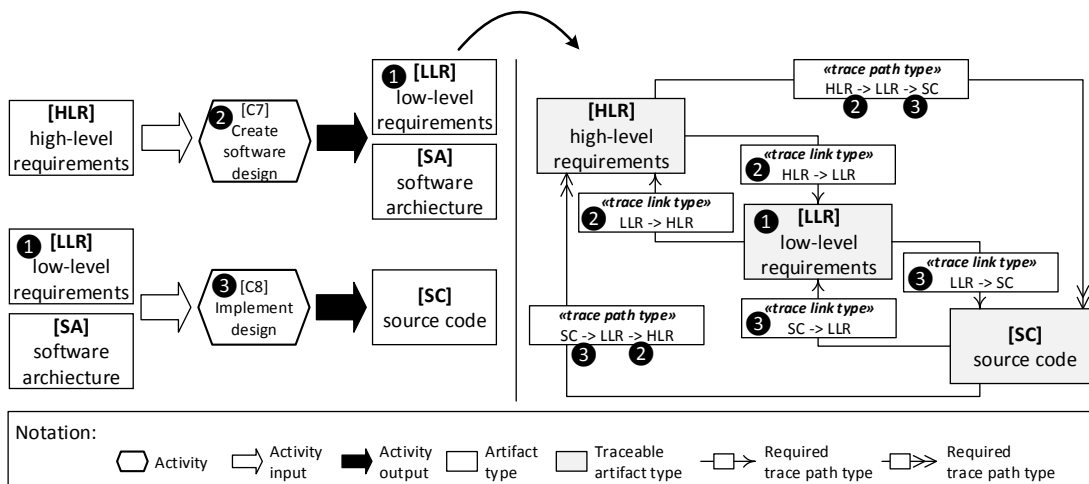


Figure 4.8.: The autopilot project: deriving required trace link types from required trace path types and software lifecycle activities

Figure 4.7 shows how required trace link types are derived for the two required trace paths for the illustrating autopilot project. The left part of the figure shows the activities C7 and C8 of the autopilot’s software lifecycle. The right part shows the autopilot’s TIM

4.3. Justifying the Purpose of Required Trace Link Types

containing three required artifact types HLR, LLR, and SC, two required trace path types HLR → LLR → SC and SC → LLR → HLR, and four required trace link types HLR → LLR, LLR → SC, LLR → HLR, and SC → LLR. The created TIM is the result of the finding that the artifact type HLR evolves to the artifact type SC through the activity sequence C7, C8. The TIM was derived in three steps:

- ❶ The activity output LLR is an intermediate artifact type of the required trace path type. Since LLR is required as an intermediated artifact type, it becomes a new traceable artifact type of the TIM.
- ❷ HLR evolves to LLR through the activity C7. Thus, a required trace link type HLR → LLR is derived for the TIM. As traceability between HLR and SC is required to be bidirectional, the reverse required trace link type LLR → HLR is derived as well.
- ❸ LLR evolves to SC through the activity C8. Thus, a required trace link type LLR → SC is derived for the TIM. The reverse required trace link type is again derived due to the bidirectional required trace path type. Additionally, the sequence of required trace link types can be explicitly specified for the required trace path types, leading to the required trace path types HLR → LLR → SC and SC → LLR → HLR.

The illustrated procedure for deriving required trace link types can be repeated if the TIM draft contains multiple required trace path types. The result of this second step is a complete TIM, containing all required trace path types, required trace link types, and traceable artifact types.

4.3. Justifying the Purpose of Required Trace Link Types

A clear justification of the purpose is needed for every required trace link type. Only if this is ensured, the derived TIM can be considered a valid target state for the assessment of the fitness for purpose of implemented traceability. If this justification of the purpose is missing a single required trace link type, the entire assessment result can be questioned, as a potentially invalid target state may have been used for assessment with respect to the fitness for purpose. Thus, this section provides a discussion on why the proposed planning approach with the resulting TIM ensures that the purpose of each containing required trace link type can be justified.

The proposed planning approach for purposed traceability consists of six consecutive steps. Each step builds upon the results of the preceding step. Following these steps in the proposed order establishes a chain of evidence from a traceability goal to the results required trace link types. The chain evidence can be traced backward, from the required trace link type to a traceability goal, to verify that each required trace link type is justified by a traceability goal. It can also be traced forward, from a traceability goal

4. Planning for Purposed Traceability

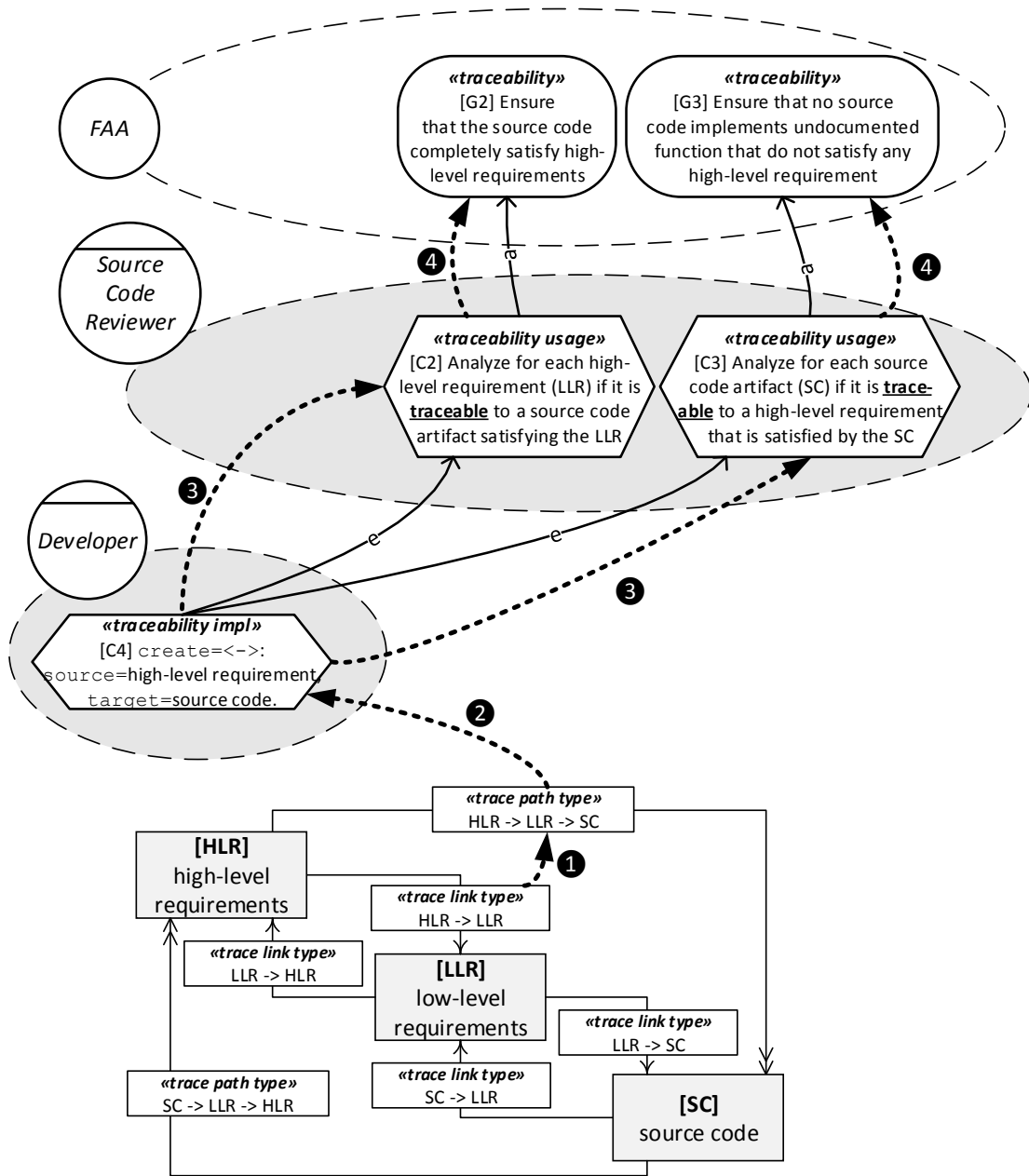


Figure 4.9.: The autopilot project: chain of evidence to justify the purpose of the required trace link type HLR -> LLR

to the required trace link types, to verify that all traceability goals have been addressed by required trace link types in the TIM. In fact, following this chain of evidence provides traceability for the required trace link types.

Figure 4.9 illustrates the chain of evidence for the required trace link type HLR -> LLR

of the autopilot example. It consists of the following steps:

- ❶ A required trace link type is part of a sequence of required trace link types that form a required trace path type (see Section 4.2.3). This part-of relationship allows the identification of the required trace path types that contain the required trace link type for which the purpose needs to be justified.
- ❷ A required trace path type is always derived from a traceability implementation activity (see Section 4.2.2). This derive dependency allows the identification of the traceability implementation activity from which the required trace path type was derived.
- ❸ A traceability implementation activity enables a traceability usage activity (see Section 4.1.5). Following this enable relationship allows to identify traceability usage activity.
- ❹ A traceability usage activity is performed to achieve a traceability goal (see Section 4.1.4). Following the achieve relationship allows to identify originating traceability goal.

It can be concluded that the traceability goals **G2** and **G3** can be identified as the purpose of the required trace link type **HLR** -> **LLR** by following the illustrated chain of evidence. This can be achieved in a similar way for all required trace link types, because all were derived with the same planning methodology. This implies that the purpose can be justified for every required trace link type of the derived TIM. Thus, it can be used as a target state to assess the fitness for purpose of a project's traceability implementation.

4.4. Summary

This chapter has presented a method for the planning of purposed traceability. It represents the first part of the proposed traceability assessment approach (see Section 3.3). Main objective of this part is to ensure purposed traceability. In this section, a summary of the presented method is provided with respect to the challenges that are related to purposed traceability (see Section 3.1.1).

As stated in Challenge 1, the planning for purposed traceability needs to be supported. The proposed approach addresses this challenge by introducing a traceability goal oriented planning methodology. As illustrated in Section 4.3, each required traceability information that is created by the proposed planning approach is justified by one or many purposes, which are represented as traceability goals.

Challenge 2 emphasized that the planning should be able to unify traceability requirements with respect to multiple purposes. As shown in Sections 4.2.2 and 4.2.3, the TIM

4. Planning for Purposed Traceability

is derived from the TRM. The TRM can contain as many traceability goals as necessary. As a consequence, the derived TIM unifies the required traceability information of all traceability goals.

Planning for purposed traceability is the first part of the proposed traceability assessment approach (see Section 3.3). The following chapter will present the second part.

5. Assessing the Fitness for Purpose of Implemented Traceability

As discussed in Section 2.3, the current state of the art mainly considers three relevant traceability lifecycle activities, namely traceability planning, traceability implementation, and traceability usage. The critique of the state of the art in Section 2.6 has identified the urgent need to verify that the implemented traceability is trustworthy. Hence, the traceability lifecycle is extended by an additional traceability verification activity. As depicted in Figure 5.1, the newly introduced traceability verification activity is driven by the output of the traceability planning activity and assesses the output of the traceability implementation activity.

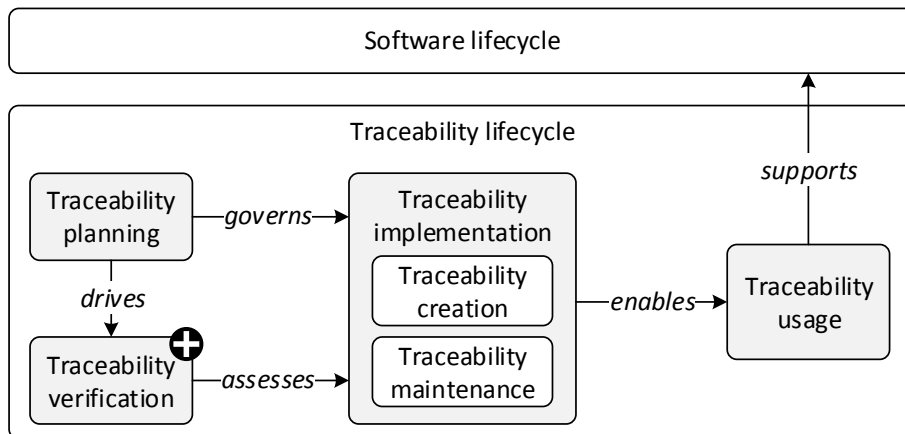


Figure 5.1.: Overview of the traceability lifecycle including the new traceability verification activity

To provide the means for the newly introduced traceability verification activity, this chapter presents an approach to assess the fitness for purpose of a project's traceability implementation. As a prerequisite, the assessment approach requires the specification of a target state for the project's traceability implementation with respect to its traceability goals. The previous chapter presented a traceability planning approach to specify this target state. The presentation of the assessment approach in this chapter is organized as follows: Section 5.1 introduces an analytical assessment model for traceability, which is

the foundation for the traceability assessment approach. Section 5.2 identifies relevant quality attributes that define the fitness for purpose of a project’s traceability implementation. Section 5.3 defines assessable traceability implementation properties with respect to the identified quality attributes that are relevant for the fitness for purpose. A comprehensive classification of traceability problems with respect to the identified quality attributes is presented in Section 5.4. The classification includes the definition of assessable traceability implementation properties that indicate these traceability problems. Dependencies among these traceability problems are discussed in Section 5.5. These problem dependencies are relevant to understand the implications of traceability assessment results. Section 5.6 discusses how the presented traceability assessment approach can be operationalized in software projects. In Section 5.7, a discussion of the presented traceability assessment approach is provided with respect to the identified challenges (see Section 3.1), which were derived from the critique of the state of the art (see Section 2.6). Section 5.7 summarizes the assessment approach with respect to the challenges that are related to trusted traceability (see Section 3.1.2).

5.1. A Traceability Assessment Model

Main goal of this approach is to assess a project’s traceability implementation for its fitness for purpose. This means that an existing traceability implementation is supposed to be assessed with respect to specific quality attributes that refer to the fitness of purpose. As a groundwork for this assessment, a Traceability Assessment Model (TAM) is proposed that defines the relevant concept for the performance of a traceability assessment. To provide a precise definition of these concepts, a meta-model is defined for the TAM. Figure 5.2 provides an overview of the defined meta-model as UML class diagram. The remainder of the section discusses each meta-model **element** in detail.

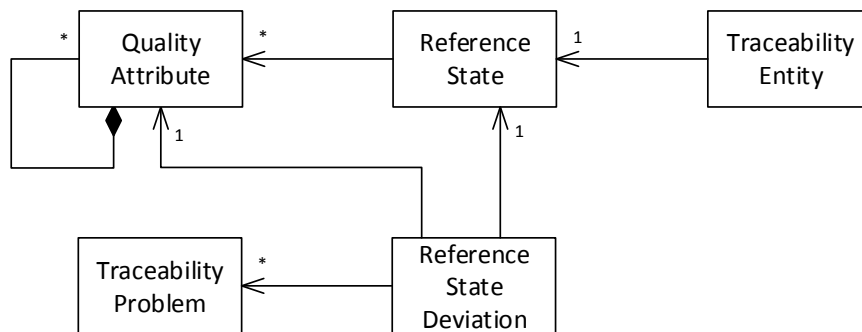


Figure 5.2.: Meta-model of the Traceability Assessment Model (TAM)

Traceability Entity. A traceability entity subsumes all elements of a project specific

traceability implementation (see Section 3.2) that is supposed to be assessed for its fitness for purpose. As depicted in Figure 5.3, the traceability entity element subsumes the traceability implementation data elements artifact, artifact type, trace link, trace link type, trace path, and trace path type.

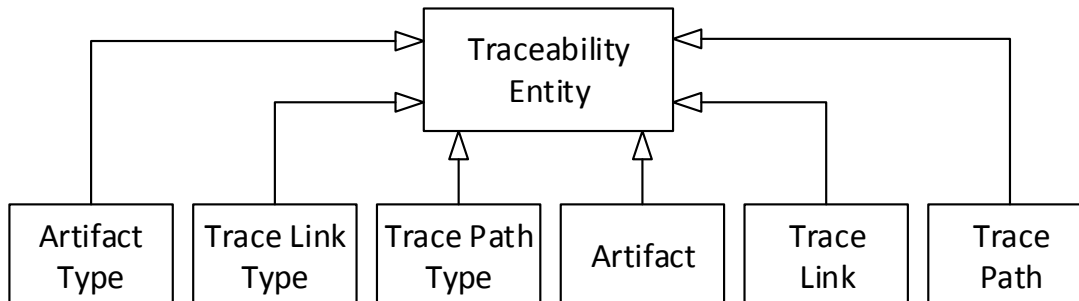


Figure 5.3.: Overview of the traceability implementation elements that are subsumed by the TAM element traceability entity

Quality Attribute. Each quality characteristic for which a project’s traceability implementation is supposed to be assessed is represented as a quality attribute. It should be noted that a quality attribute can represent very abstract quality characteristics as well as more concrete sub-characteristics. Therefore, the TAM meta-model specifies that a quality attribute can be composed of other quality attributes. Defining quality attribute hierarchies is a common approach in existing software product quality models [Boehm 1978; ISO 25010:2011].

Reference State. Each quality attribute needs to be mapped to properties of a project’s traceability implementation that can be assessed for this quality. For this purpose, a reference state is defined for each entity of a traceability implementation. It specifies an assessable property of a traceability entity that corresponds to a specific quality attribute. To make this correspondence to a quality attributes explicit, each reference state is mapped to one or many quality attributes.

Reference State Deviation. A project specific traceability implementation may deviate from this reference state. Recent studies [Mäder et al. 2013; Rempel et al. 2013; Rempel et al. 2014] have shown that these deviations are the rule rather than an exception. The existence of such deviations from the reference state indicate a problem with respect to the quality attribute that is related to the reference state. The reference state deviation element represents an assessable property of a traceability entity that indicates a traceability problem.

Traceability Problem. The aforementioned problems, which are indicated by a refer-

ence state deviation, are represented by the element traceability problem. Traceability problems are explicitly associated to the reference state deviation that indicates the problem. A reference state deviation may indicate one or many traceability problems.

5.2. Quality Attributes of a Purposed Traceability Implementation

This section identifies and defines the relevant quality attributes, which are expected from a software project’s traceability implementation to be fit for purpose. The ISO 25010 standard on Systems and software Quality Requirements and Evaluation (SQuaRE) defines a product quality model that consists of eight high-level quality attributes such as *functional suitability*, *performance efficiency*, *compatibility*, *usability*, *reliability*, *security*, *maintainability*, and *portability* [ISO 25010:2011]. Since the presented approach focuses solely on assessing the fitness for purpose, functional suitability is the relevant high-level quality attribute. Functional suitability is defined as the “*degree to which a product or system provides functions that meet stated and implied needs when used under specified conditions*” [ISO 25010:2011]. As illustrated in Figure 5.4, this high-level quality attribute is composed of the three sub-characteristics *completeness*, *appropriateness*, and *correctness*.

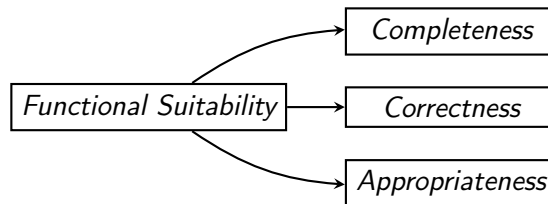


Figure 5.4.: Quality tree of the high-level attribute functional suitability

The assessment approach focuses on assessing the quality of traceability implementations only rather than the entire software system. Hence, the definitions of completeness, appropriateness, and correctness, as provided in [ISO 25010:2011], are adapted in this thesis for the specific scope of traceability implementation assessments.

Definition 6 (Completeness). *It refers to the degree to which a project’s traceability implementation data cover all required traceability information as specified in the TIM.*

Definition 7 (Appropriateness). *It refers to the degree to which a project’s traceability implementation data facilitate the achievement of the traceability goals that are specified within the TRM.*

Definition 8 (Correctness). *It refers to the degree of precision to which a project provides traceability implementation data with respect to the required traceability information.*

Hence, to assess a project’s traceability implementation for its fitness for purpose, the respective traceability implementation data need to be assessed with respect to the quality attributes completeness, appropriateness and correctness.

5.3. Assessable Traceability Implementation Properties with Respect to Purposed Traceability

The quality attributes completeness, appropriateness, and correctness are abstract concepts, which cannot be directly observed in a project’s traceability implementation data. Thus, this section defines the properties of traceability implementation data that are related to the quality attributes. For each traceability implementation data element, a generalized reference state is defined to specify what is required so that any instance of the respective traceability implementation data element is complete, appropriate, and correct. Additionally, a formal expression is provided for each reference state. This formal expression can be used to determine whether or not a specific traceability implementation data instance fulfills the reference state requirements. An illustrating example is provided for each traceability implementation data element.

Artifact Type. A fundamental assumption for the completeness of traceability is that all artifact types that are required to be traceable are implemented. For the appropriateness of traceability it is necessary that all implemented artifact types are required. The correctness of traceability is not directly affected by artifact types. Reference state: For each required artifact type that is at least a source or a target artifact type of one required trace link type in the TIM, there exists a corresponding artifact type within the TID. Each implemented artifact type should correspond to one required artifact type of the TIM. Formal expression: $\forall r \in \mathcal{R}_{\mathcal{A}} \forall i \in I_{\mathcal{A}} [implements^{-1}(r) \in I_{\mathcal{A}} \wedge implements(i) \in \mathcal{R}_{\mathcal{A}}]$. Example: Figure 5.5 sketches an example of valid artifact type implementations

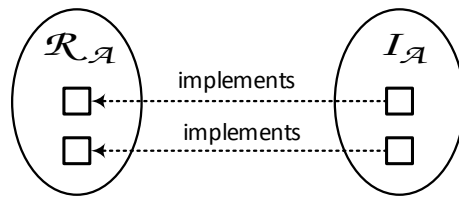


Figure 5.5.: Example of valid artifact type implementations

as Venn diagram. The shown example is valid because all required artifact types are implemented (complete) and all implemented artifact types are required (appropriate).

Trace Link Type. For the completeness of traceability it is also required that all required trace link types are implemented. For the appropriateness of traceability it is

5. Assessing the Fitness for Purpose of Implemented Traceability

necessary that all implemented trace link types are required. For the correctness of the traceability it is necessary that all required direct trace link types are implemented by a trace link type. Reference state: For each required trace link type in the TIM there exists a corresponding trace link type within the TID that implements the corresponding required trace link type. Each trace link type that is implemented in TID corresponds to one required trace link type of the TIM that it implements. Formal expression: $\forall r \in \mathcal{R}_{\mathcal{L}} \forall i \in I_{\mathcal{L}} [implements^{-1}(r) \in I_{\mathcal{L}} \wedge implements(i) \in \mathcal{R}_{\mathcal{L}}]$. Example: Figure 5.6 sketches an example of a valid trace link type implementation as Venn diagram. The shown example is valid because all required trace link types are implemented (complete), all implemented trace link types are required (appropriate), all required direct traceability relations are implemented by a trace link type (correct).

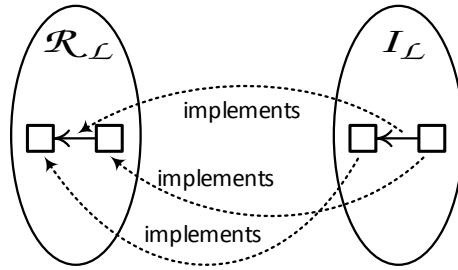


Figure 5.6.: Example of a valid trace link type implementation

Trace Path Type. For the completeness of traceability it is also required that all required trace path types are implemented. For the appropriateness of traceability it is necessary that all implemented trace path types are required. For the correctness of the traceability it is necessary that all required path types are implemented by a trace path type. Reference state: For each required trace path type in the TIM there exists a corresponding trace path type within the TID that implements the required trace path type. Each trace path type that is implemented in TID corresponds to one required trace path type of the TIM that it implements. Formal expression: $\forall r \in \mathcal{R}_{\mathcal{P}} \forall i \in I_{\mathcal{P}} [implements^{-1}(r) \in I_{\mathcal{P}} \wedge implements(i) \in \mathcal{R}_{\mathcal{P}}]$. Example: Figure 5.7 sketches an example of a valid trace path type implementation as Venn diagram. The shown example is valid because all the required transitive traceability relations are implemented (complete), all implemented trace path types are required (appropriate), all required path types are implemented by a trace path type (correct).

Artifact. Another fundamental assumption for the completeness of traceability is that all implemented artifact types are instantiated by concrete artifacts. For the appropriateness of traceability it is necessary that all implemented artifacts are an instance of an implemented artifact type. The correctness of traceability is not directly affected

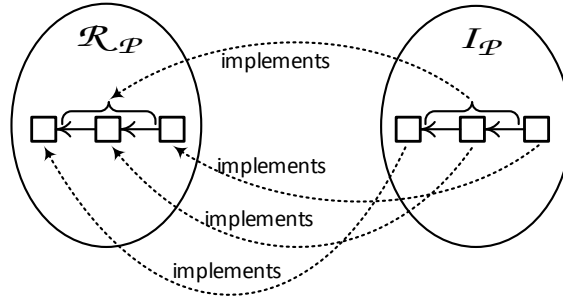


Figure 5.7.: Example of a valid trace path type implementation

by artifacts. Reference state: For each implemented artifact type there exists one or many artifacts that are instances of the implemented artifact type. Each implemented artifact is an instance of an implemented artifact type in the TID. Formal expression: $\forall i_t \in I_{\mathcal{A}} \forall i_i \in I_A [instances(i_t) \neq \emptyset \wedge instances^{-1}(i_i) \in I_{\mathcal{A}}]$. Example: Figure 5.8 sketches an example of valid artifact implementations as Venn diagram. The shown example is valid because all required artifact types are implemented by artifact types that have artifact instances (complete) and all implemented artifacts are an instance of a required artifact type and therefore required (appropriate).

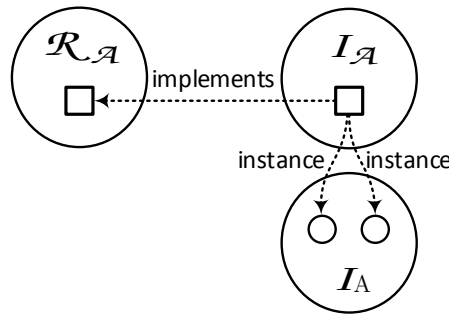


Figure 5.8.: Example of valid artifact implementation

Trace Link. For the completeness of traceability it is also required that each artifact that is an instance of an artifact type belonging to a trace link type is part of a trace link that instantiate this trace link type. For the appropriateness of traceability it is necessary that all implemented trace links are an instance of an implemented trace link type. For the correctness of the traceability it is necessary that the implemented trace link types are instantiated by trace paths. Reference state: For each artifact that is an instance of an artifact type as either source or target of a trace link type t in the TID there exists a trace link that is an instance of this t . For each implemented trace link l in the TID there exists a trace link type that is instantiated by l . For-

5. Assessing the Fitness for Purpose of Implemented Traceability

mal expression: $\forall a_s \subseteq instance(source(I_L)) \exists l_s \in I_L [l_s \in source^{-1}(instance^{-1}(a_s))] \wedge \forall a_t \subseteq instance(target(I_L)) \exists l_t \in I_L [l_t \in target^{-1}(instance^{-1}(a_t))]$. Example: Figure 5.9 sketches an example of a trace link implementation as Venn diagram. The shown example is valid, because for all artifacts a trace link is available that are instances of the trace link type to which the artifacts belong (complete), all implemented trace links are instances of a trace link type to which the related artifacts belong (appropriate), all trace link types are instantiated by a trace link (correct).

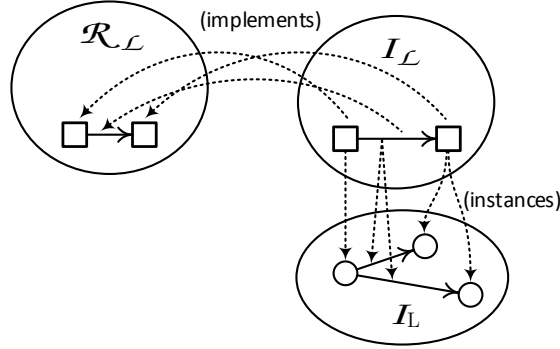


Figure 5.9.: Example of a valid trace link implementations

Trace Path. For the completeness of traceability it is required that each artifact being an instance of an artifact type that belongs to a trace path type is part of a trace path to instantiate this trace path type. For the appropriateness of traceability it is necessary that all implemented trace paths are an instance of an implemented trace path type. For the correctness of the traceability it is necessary that the implemented trace path types are instantiated by trace paths. Reference state: For each artifact that is an instance of an artifact type being either a source or a target of a trace path type t in the TID there exists a trace path that is an instance of this t . For each implemented trace path p in the TID there exists a trace path type that is instantiated by p . Formal expression: $\forall a_s \subseteq instance(source(I_P)) \exists p_s \in I_P [p_s \in source^{-1}(instance^{-1}(a_s))] \wedge \forall a_t \subseteq instance(target(I_P)) \exists p_t \in I_P [p_t \in target^{-1}(instance^{-1}(a_t))]$. Example: Figure 5.10 sketches an example of a valid trace path implementation as Venn diagram. The shown example is valid, because for all artifacts a trace path is available that is an instances of the trace path type to which the artifacts belong (complete), all implemented trace paths are instances of a trace path type to which the related artifacts belong (appropriate), all trace path types are instantiated by a trace path (correct).

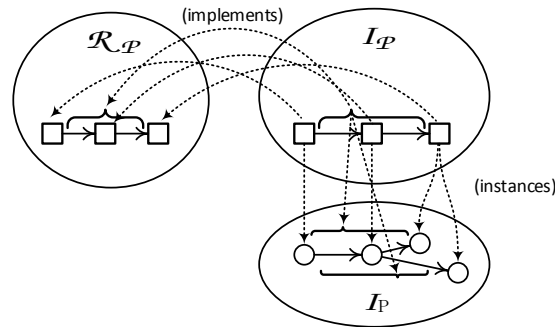


Figure 5.10.: Example of a valid trace path implementations

5.4. Traceability Problems

Implemented traceability data can deviate from the reference state. These reference state deviations indicate traceability problems with respect to the fitness for purpose of a project's traceability implementation. This section provides a comprehensive classification of traceability problems. The discussion of traceability problems is organized by the quality attributes completeness (see Section 5.4.1), appropriateness (see Section 5.4.2), and correctness (5.4.3).

5.4.1. Problems Related to the Completeness

This section presents traceability problems that are related to the completeness of traceability implementation data. For each traceability implementation data element, the traceability problem is discussed. Additionally, a generalized reference state deviation is defined that indicate the existence of this problem. Additionally, a formal expression is provided for each reference state deviation. This formal expression can be used to determine unambiguously whether or not a specific traceability implementation data instance deviates from its reference state. For each traceability problem, an illustrating example is provided.

Missing Artifact Type (\mathcal{M}_a). A traceability implementation is incomplete if a required artifact class is not implemented. This implies that the implementation of an artifact type is missing. Reference state deviation: There exists a required artifact type that is a source or a target artifact type of a required trace link type in the TIM for which no corresponding artifact type is implemented within the TID. Formal expression: $\exists a \in \mathcal{R}_a [implements^{-1}(a) = \emptyset] \implies a \in \mathcal{M}_a$. Example: Figure 5.11 sketches an example of an incomplete traceability implementation that misses an artifact type. The first of the required artifact types that is highlighted with a question-mark is not implemented by an artifact type.

5. Assessing the Fitness for Purpose of Implemented Traceability

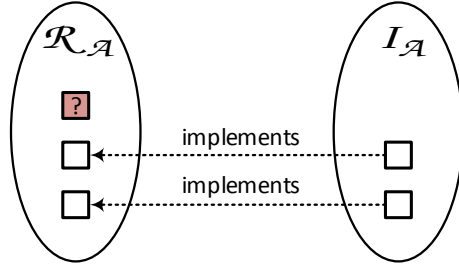


Figure 5.11.: Example of an incomplete traceability implementation that misses artifact type

Missing Trace Link Type (\mathcal{M}_L). A traceability implementation is incomplete if a required trace link type is not implemented. This implies that the implementation of a trace link type is missing. Reference state deviation: There exists a required trace link type in the TIM for which no corresponding trace link type is implemented within the TID. Formal expression: $\exists l \in \mathcal{R}_L [implements^{-1}(l) = \emptyset] \implies l \in \mathcal{M}_L$. Example: Figure 5.12 sketches an example of an incomplete traceability implementation that misses a trace link type. The required trace link type that is highlighted with a question-mark is not implemented by a trace link type.

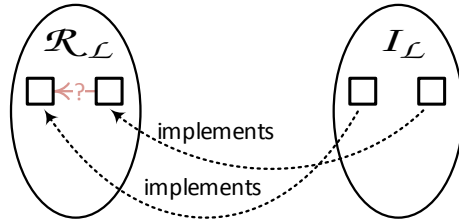


Figure 5.12.: Example of an incomplete traceability implementation that misses a trace link type

Missing Trace Path Type (\mathcal{M}_P). A traceability implementation is incomplete if a required trace path type is not implemented. This implies that the implementation of a trace path type is missing. Reference state deviation: There exists a required trace path type in the TIM for which no corresponding trace path type is implemented within the TID. Formal expression: $\exists p \in \mathcal{R}_P [implements^{-1}(p) = \emptyset] \implies p \in \mathcal{M}_P$. Example: Figure 5.13 sketches an example of an incomplete traceability implementation that misses a trace path type. The required trace path type that is highlighted with a question-mark is not implemented by a trace path type.

Missing Artifact (\mathcal{M}_A). A traceability implementation is incomplete if an artifact type is not instantiated by an artifact. This implies that the implementation of an

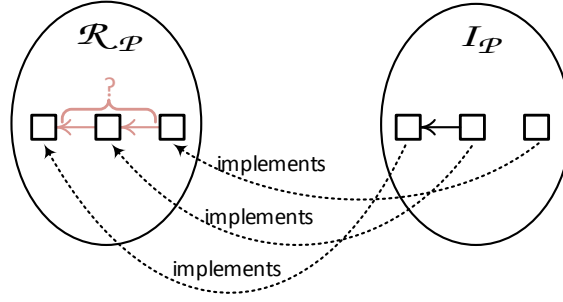


Figure 5.13.: Example of an incomplete traceability implementation that misses a trace path type

artifact is missing. Reference state deviation: There exists an artifact type in the TID for which no artifact is implemented that is an instances of the artifact type. Formal expression: $\exists a \in I_{\mathcal{A}}[instances(a) = \emptyset] \implies a \in \mathcal{M}_{\mathcal{A}}$. Example: Figure 5.14 sketches an example of an incomplete traceability implementation that misses an artifact. The artifact type that is highlighted with a question-mark is not implemented by any artifact instance.

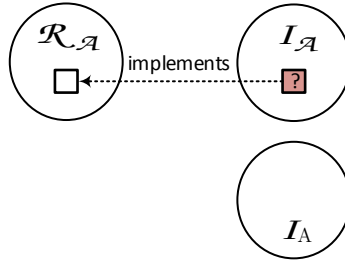


Figure 5.14.: Example of an incomplete traceability implementation that misses artifacts

Missing Trace Link (\mathcal{M}_L). A traceability implementation is incomplete if an artifact that is an instance of an artifact type belonging to a trace link type is not part of a trace link that instantiate this trace link type. This implies that the implementation of a trace link is missing. Reference state deviation: There exists an artifact which is an instance of an artifact type that is either a source or a target of a trace link type t in the TID for which no trace link is implemented which is an instance of this t . Formal expression: $\exists a_s \subseteq instance(source(I_L)) \neg \exists l_s \in I_L[l_s \in source^{-1}(instance^{-1}(a_s))] \implies l_s \in \mathcal{M}_L$. $\exists a_t \subseteq instance(target(I_L)) \neg \exists l_t \in I_L[l_t \in target^{-1}(instance^{-1}(a_t))] \implies l_t \in \mathcal{M}_L$. Example: Figure 5.15 sketches an example of an incomplete traceability implementation that misses a trace link. The artifact that is highlighted with a question-mark is an instance of an artifact type that is the target of a trace link type. However, this highlighted artifact is not a target of a trace link that instantiates the trace link

type.

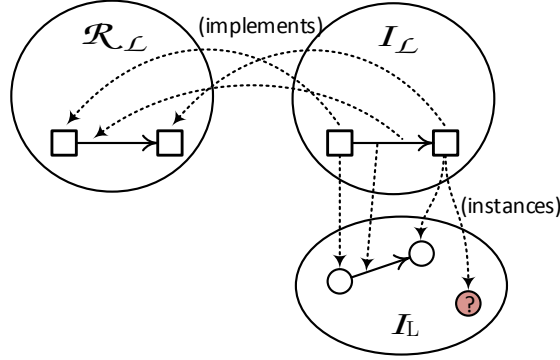


Figure 5.15.: Example of an incomplete traceability implementation that misses trace links

Missing Trace Path (\mathcal{M}_P). A traceability implementation is incomplete if an artifact that is an instance of an artifact type belonging to a trace path type which is not part of a trace path that instantiates this trace path type. This implies that the implementation of a trace path is missing. Reference state deviation: There exists an artifact which is an instance of an artifact type that is either a source or a target of a trace path type t in the TID for which no trace path is implemented which is an instance of this t . Formal expression: $\exists a_s \subseteq instance(source(I_p)) \neg \exists p_s \in I_p [p_s \in source^{-1}(instance^{-1}(a_s))] \implies p_s \in \mathcal{M}_P$. $\exists a_t \subseteq instance(target(I_p)) \neg \exists p_t \in I_p [p_t \in target^{-1}(instance^{-1}(a_t))] \implies p_t \in \mathcal{M}_P$. Example: Figure 5.16 sketches an example of an incomplete traceability implementation that misses a trace path. The artifact that is highlighted with a question-mark is an instance of an artifact type that is the target of a trace path type. However, this highlighted artifact is not a target of a trace path that instantiates the trace path type.

5.4.2. Problems Related to the Appropriateness

This section presents traceability problems that are related to the appropriateness of traceability implementation data. The presentation of the traceability problems is organized as in the previous section.

Superfluous Artifact Type (\mathcal{S}_A). A traceability implementation is not appropriate if an implemented artifact type does not correspond to a required artifact type. This implies that the implementation of this artifact type is superfluous. Reference state deviation: There exists an artifact type that is implemented in TID that does not correspond to a required artifact type of the TIM. Formal expression: $\exists a \in I_A [implements(a) = \emptyset] \implies a \in \mathcal{S}_A$. Example: Figure 5.17 sketches an example of an inappropriate

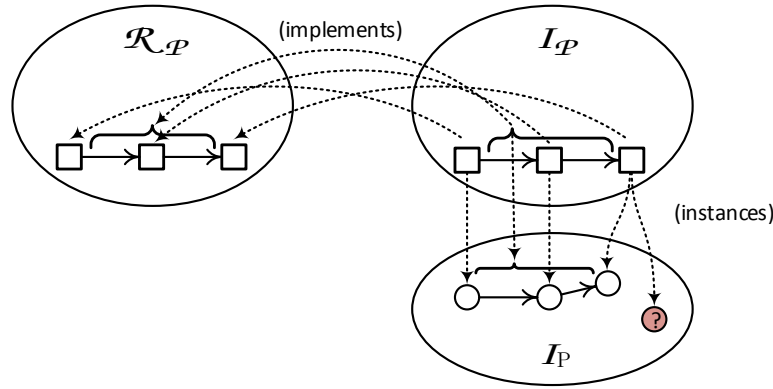


Figure 5.16.: Example of an incomplete traceability implementation that misses trace paths

traceability implementation, which contains a superfluous artifact type. The artifact type that is highlighted with a question-mark does not correspond to a required artifact type.

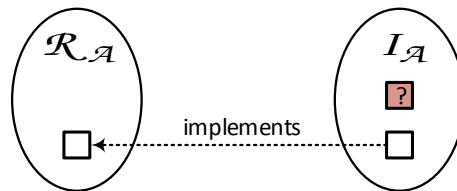


Figure 5.17.: Example of an inappropriate traceability implementation that contains a superfluous artifact type

Superfluous Trace Link Type (\mathcal{S}_L). A traceability implementation is not appropriate if an implemented trace link type does not correspond to a required trace link type. This implies that the implementation of this trace link type is superfluous. Reference state deviation: There exists a trace link type that is implemented in the TID that does not correspond to a required trace link type of the TIM. Formal expression: $\exists l \in I_L [implements(l) = \emptyset] \implies l \in \mathcal{S}_L$. Example: Figure 5.18 sketches an example of an inappropriate traceability implementation, which contains a superfluous trace link type. The trace link type that is highlighted with a question-mark does not correspond to a required trace link type.

Superfluous Trace Path Type (\mathcal{S}_P). A traceability implementation is not appropriate if an implemented trace path type does not correspond to a required trace path type. This implies that the implementation of this trace path type is superfluous. Reference state deviation: There exists a trace path type that is implemented in TID and that

5. Assessing the Fitness for Purpose of Implemented Traceability

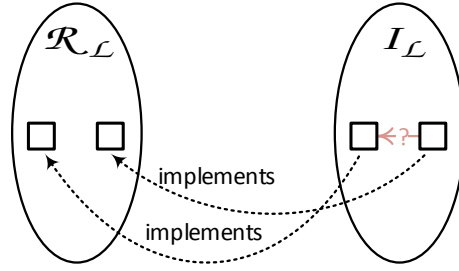


Figure 5.18.: Example of an inappropriate traceability implementation that contains a superfluous trace link type

does not correspond to one required trace path type of the TIM. Formal expression: $\exists p \in I_{\mathcal{P}}[\text{implements}(p) = \emptyset] \implies p \in \mathcal{S}_{\mathcal{P}}$. Example: Figure 5.19 sketches an example of an inappropriate traceability implementation, which contains a superfluous trace path type. The trace path type that is highlighted with a question-mark does not correspond to a required trace path type.

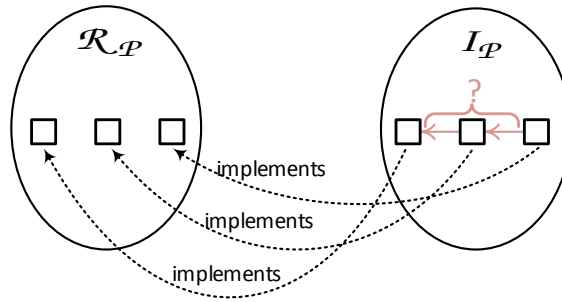


Figure 5.19.: Example of an inappropriate traceability implementation that contains a superfluous trace path type

Superfluous Artifact (\mathcal{S}_A). A traceability implementation is not appropriate if an implemented artifact is an instance of an artifact type that does not correspond to a required artifact type. This implies that the implementation of this artifact is superfluous. Reference state deviation: There exists an artifact that is implemented in that TID and that is an instance of an artifact type. However, this artifact type does not correspond to a required artifact type in the TIM. Formal expression: $\exists a \in I_A[\text{implements}(\text{instance}^{-1}(a)) = \emptyset] \implies a \in \mathcal{S}_A$. Example: Figure 5.20 sketches an example of an inappropriate traceability implementation, which contains a superfluous artifact. All artifacts that are highlighted with a question-mark are an instance of an artifact type, which is also highlighted with a question-mark and that does not correspond to a required artifact type.

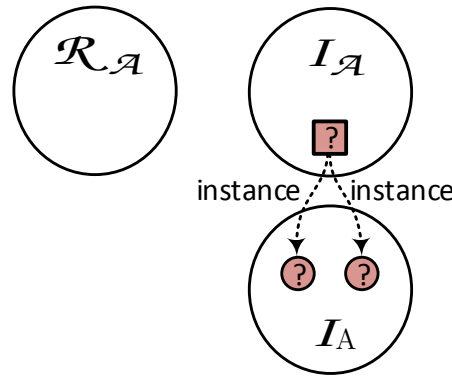


Figure 5.20.: Example of an inappropriate traceability implementation that contains superfluous artifacts

Superfluous Trace Link (\mathcal{S}_L). A traceability implementation is not appropriate if an implemented trace link is an instance of a trace link type, which does not correspond to a required trace link type. This implies that the implementation of this trace link is superfluous. Reference state deviation: There exists a trace link that is implemented in the TID and that is an instance of a trace link type, which does not correspond to a required trace link type of the TIM. Formal expression: $\exists l \in I_L[\text{implements}(\text{instance}^{-1}(l)) = \emptyset] \implies l \in \mathcal{S}_L$. Example: Figure 5.21 sketches an example of an inappropriate traceability implementation, which contains superfluous trace links. All trace links that are highlighted with a question mark are an instance of the trace link type that is also highlighted with a question-mark, which does not correspond to a required trace link type.

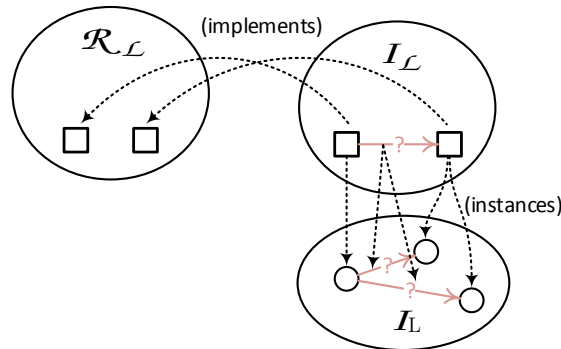


Figure 5.21.: Example of an inappropriate traceability implementation that contains superfluous trace links

Superfluous Trace Path (\mathcal{S}_p). A traceability implementation is not appropriate if an implemented trace path is an instance of a trace path type that does not correspond

5. Assessing the Fitness for Purpose of Implemented Traceability

to a required trace path type. This implies that the implementation of this trace path is superfluous. Reference state deviation: There exists a trace path implemented in TID that is an instance of a trace path type, which does not correspond to a required trace path type of the TIM. Formal expression: $\exists p \in I_P[\text{implements}(\text{instance}^{-1}(p)) = \emptyset] \implies p \in \mathcal{S}_P$. Example: Figure 5.22 sketches an example of an inappropriate traceability implementation, which contains superfluous trace paths. All trace paths that are highlighted with a question mark are an instance of the trace path type that is also highlighted with a question-mark, which does not correspond to a required trace path type.

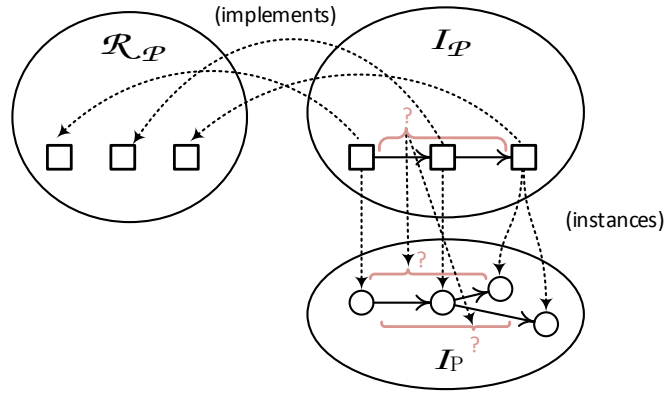


Figure 5.22.: Example of an inappropriate traceability implementation that contains superfluous trace paths

5.4.3. Problems Related to the Correctness

This section presents traceability problems that are related to the correctness of traceability implementation data. The presentation of the traceability problems is organized as in the previous section. As discussed in Section 5.3, the correctness of traceability is not directly affected by artifacts or artifact types. Accordingly, this section does not define correctness problems for these two traceability implementation data elements.

Wrong Trace Link Type (\mathcal{W}_L). A traceability implementation is incorrect if a required trace path type is implemented by a trace link type. This implies that the implementation of the trace link type is wrong. Reference state deviation: There exists a required trace path type in the TIM for which a corresponding trace link type is implemented within the TID. Formal expression: $\exists l \in I_L[\text{implements}(l) \in \mathcal{R}_P] \implies l \in \mathcal{W}_L$. Example: Figure 5.23 sketches an example of an incorrect traceability implementation that contains a wrong trace link type. The trace link type that is highlighted with a question mark implements a required trace path type.

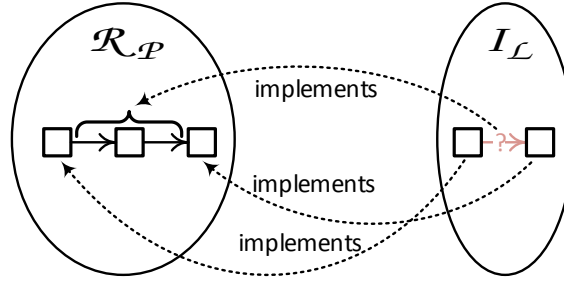


Figure 5.23.: Example of an incorrect traceability implementation that contains a wrong trace link type

Wrong Trace Path Type (\mathcal{W}_P). An implemented trace path type can be wrongly implemented in two ways. A traceability implementation is incorrect if either a required trace link type is implemented by a trace path type or if a required trace path type is implemented by a trace path type where at least one intermediate artifact type at the n -th position of the path does not implement the corresponding required artifact type at the n -th position of the required trace path type. Both cases imply that the implementation of a TRACE PATH TYPE is wrong. Reference state deviation-A: There exists a trace path type within the TID that implements a required trace link type in the TIM. Formal expression-A: $\exists p \in I_p[\text{implements}(p) \in \mathcal{R}_L] \implies p \in \mathcal{W}_P$. Example-A: Figure 5.24 sketches an example of the first alternative of an incorrect traceability implementation that contains a wrong trace path type. The trace path type that is highlighted with a question mark implements a required trace link type. Reference state deviation-B: There exists a trace path type within the TID that implements a required trace path type of the TIM. For the latter one, there exists a trace link type that is a step of the trace path type but does not implement any step of the required trace path type. Formal expression-B: $\exists p \in I_p \exists l \in \text{steps}(p)[l \notin \text{steps}(\text{implements}(p))] \implies p \in \mathcal{W}_P$. Example-B: Figure 5.25 sketches an example of the second alternative of an incorrect traceability implementation that contains a wrong trace path type. The trace path type that is highlighted with a question-mark contains two steps that do not implement the corresponding step of the implemented required trace path type.

Wrong Trace Link (\mathcal{W}_L). A traceability implementation is incorrect if a wrong trace link type is instantiated by trace links. This implies that the created trace link instances are wrong too. Reference state deviation: There exists a trace link within the TID that instantiates a trace link type that implements a required trace path type. Formal expression: $\exists l \in I_L[\text{implements}(\text{instance}^{-1}(l)) \in \mathcal{R}_P] \implies l \in \mathcal{W}_L$. Example: Figure 5.26 sketches an example of an incorrect traceability implementation that contains wrong trace links. The trace links that are highlighted with a question mark, are instances of

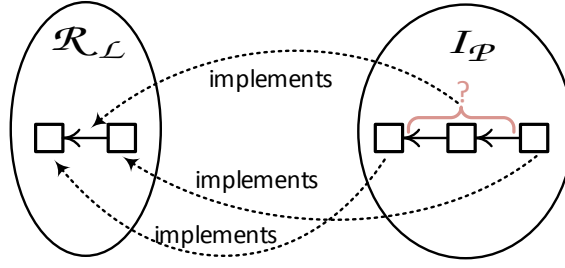


Figure 5.24.: Example of an incorrect traceability implementation that contains a wrong trace path type

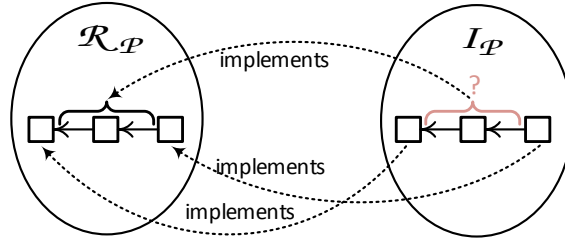


Figure 5.25.: Example of an inappropriate traceability implementation that contains superfluous trace paths

a trace link type that implements a required trace path type.

Wrong Trace Path (\mathcal{W}_P). A traceability implementation is incorrect if a wrong trace path type is instantiated by trace paths. This implies that the created trace path instances are wrong, too. Since trace path types can be wrongly implemented in two ways, there are also two alternative cases for wrongly implemented trace links. Reference state deviation-A: There exists a trace path that is an instance of a trace path type within the TID that implements a required trace link type of the TIM. Formal expression-A: $\exists p \in I_P[\text{implements}(\text{instance}^{-1}(p)) \in \mathcal{R}_L] \implies p \in \mathcal{W}_P$. Example-A: Figure 5.27 sketches an example of the first alternative of an incorrect traceability implementation that contains a wrong trace path. The trace paths that are highlighted with a question mark are instances of a trace path type that implements a required trace link type. Reference state deviation-B: There exists a trace path that is an instance of a trace path type within the TID, which implements a required trace path type of the TIM. For this trace path type there exists a trace link type that is a step of the trace path type but does not implement any step of the required trace path type. Formal expression-B: $\exists p \in I_P \exists l \in \text{steps}(p)[l \notin \text{steps}(\text{implements}(\text{instance}^{-1}(p)))] \implies p \in \mathcal{W}_P$. Example-B: Figure 5.28 sketches an example of the second alternative of an incorrect

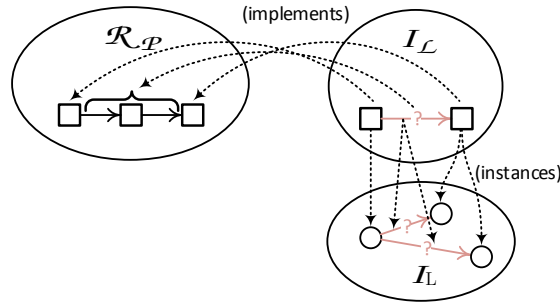


Figure 5.26.: Example of an incorrect traceability implementation that contains wrong trace links

traceability implementation that contains a wrong trace path. The trace paths that are highlighted with a question-mark are instances of a trace path type. This trace path contains two steps that do not implement the steps of the corresponding required trace path type.

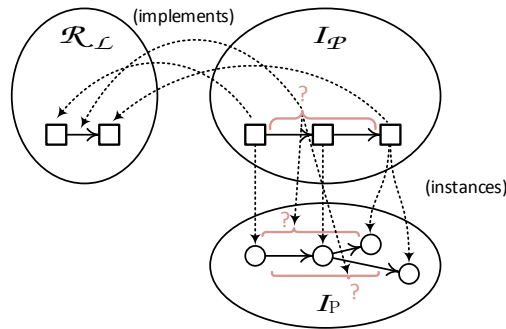


Figure 5.27.: Example of an inappropriate traceability implementation that contains superfluous trace paths

5.5. Dependencies Among the Traceability Problems

There exist dependencies among the traceability problems. These dependencies result from the fact that the traceability implementation data itself are interdependent, as discussed in Section 3.2. Knowing them can help to understand the implications of traceability assessment results. The top part of Figure 5.29 shows the dependencies among the traceability implementation data as dependency graph. The bottom part of Figure 5.29 shows the derived dependencies among the traceability problems as dependency graph. Each derived traceability problem is marked with a number. In the remainder of this section, each derivation step is discussed.

5. Assessing the Fitness for Purpose of Implemented Traceability

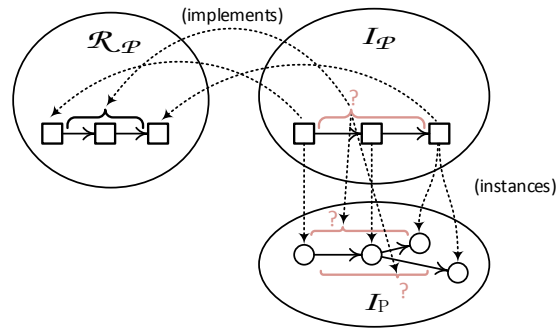


Figure 5.28.: Example of an inappropriate traceability implementation that contains superfluous trace paths

❶ A trace path type consists of a sequence of steps. Each step is represented by a trace link type. Due to this dependency, the existence of a trace path type related problem, such as a missing trace path type, superfluous trace path type, or wrong trace path type, implies for each trace link type that is a step the trace path type the existence of a corresponding missing trace link type, superfluous trace link type, or wrong trace link type problem.

❷ A trace path type is instantiated by trace path elements. Due to this dependency, the existence of a trace path type related problem, such as a missing trace path type, superfluous trace path type, or wrong trace path type, implies for each trace path that is an instance of the trace path type the existence of a corresponding missing trace path, superfluous trace path, or wrong trace path problem.

❸ A trace link type consists of a source and a target artifact type. Due to this dependency, the existence of an artifact type related problem, such as a missing artifact type or superfluous artifact type implies for the trace link type that consists of this artifact type the existence of a corresponding missing trace link type or superfluous trace link type problem.

❹ A trace link type is instantiated by trace link elements. Due to this dependency, the existence of a trace link type related problem, such as a missing trace link type, superfluous trace link type, or wrong trace link type implies for each trace link that is an instance of the trace link type the existence of a corresponding missing trace link, superfluous trace link, or wrong trace link problem.

❺ A trace path consists of a sequence of steps. Each step is represented by a trace link. Due to this dependency, the existence of a trace path related problem, such as a missing trace path, superfluous trace path, or wrong trace path, implies for each trace link that is a step of the trace path the existence of a corresponding missing trace link, superfluous trace link, or wrong trace link problem.

5.5. Dependencies Among the Traceability Problems

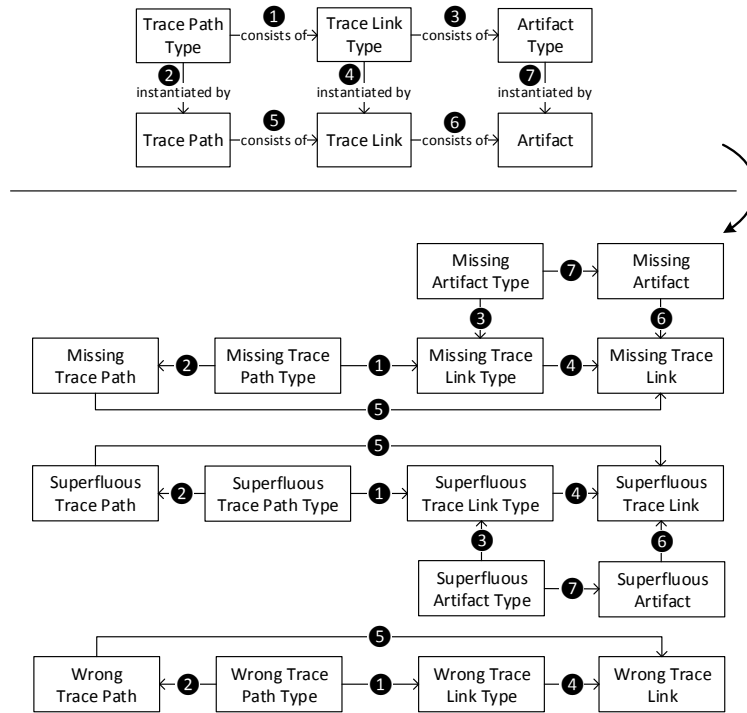


Figure 5.29.: Dependencies among the traceability implementation data and the derived dependencies among the traceability problems

⑥ A trace link consists of a source and a target artifact. Due to this dependency, the existence of an artifact related problem, such as a missing artifact or superfluous artifact, implies for the trace link that consists of this artifact the existence of a corresponding missing trace link or superfluous trace link problem.

⑦ An artifact type is instantiated by artifact elements. Due to this dependency, the existence of an artifact type related problem, such as a missing artifact type or superfluous artifact type, implies for each artifact that is an instance of the artifact type the existence of a corresponding missing artifact or superfluous artifact problem.

As envisioned in the beginning of this section, knowing these dependencies can be helpful for the interpretation of the assessment results. The existence of traceability problems that are related to an artifact type, trace link type or trace path type have typically a much higher impact, because they automatically imply the existence of this problem in all the respective instances as well. For example, a project with thousands of artifacts that are instances of a trace link type's source artifact type, the assessment result that this trace link type is missing would imply that thousands of trace links are missing as well. Thus, problems related to the elements artifact type, trace link type, or trace path type can be used as a leading indicator to detect critical areas of the project with respect to traceability. Problems related to the elements artifact, trace link and

5. Assessing the Fitness for Purpose of Implemented Traceability

trace path can then be used to draw very detailed conclusions with respect to particular artifacts.

5.6. Performing a Traceability Assessment

The proposed TAM provides means to assess the fitness for purpose of a project's traceability implementation. This section provides a discussion how the proposed TAM can be operationalized. This discussion assumes that a project specific TIM has been created already (see Section 4) and can be used as a target state for the traceability assessment.

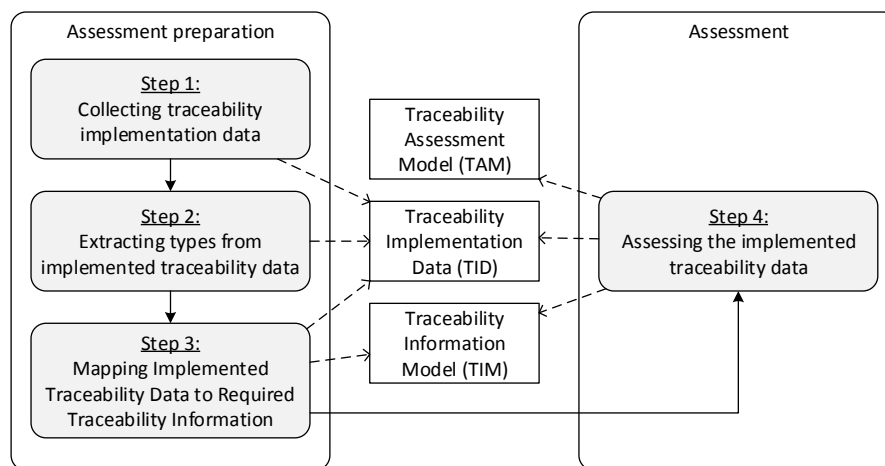


Figure 5.30.: Overview of the three preparation steps (step 1-3) and the one assessment step (step 4) that are required to perform a traceability assessment

As depicted in Figure 5.30, performing a traceability assessment consists of three TID preparation steps (step 1-3) and one TID assessment steps (step 4). In step 1, the project specific traceability information data are collected (see Section 5.6.1). Step 2 extracts type information from the collected traceability implementation data (see Section 5.6.2). Step 3 maps the extracted types to the corresponding required traceability information (see Section 5.6.3). In Step 4, the TAM is applied to the prepared TID in order to assess their fitness for purpose.

5.6.1. Step 1: Collecting Traceability Implementation Data

The first step towards assessing the quality of a traceability implementation is to collect all artifacts and trace links that existing in the software development project. This step depends on the electronic formats and the tooling to manage artifacts and trace links.

In Figure 5.31, an excerpt of the software requirements specification and the software verification plan of the open source project [TOPCASE-SAM 2015] is shown.

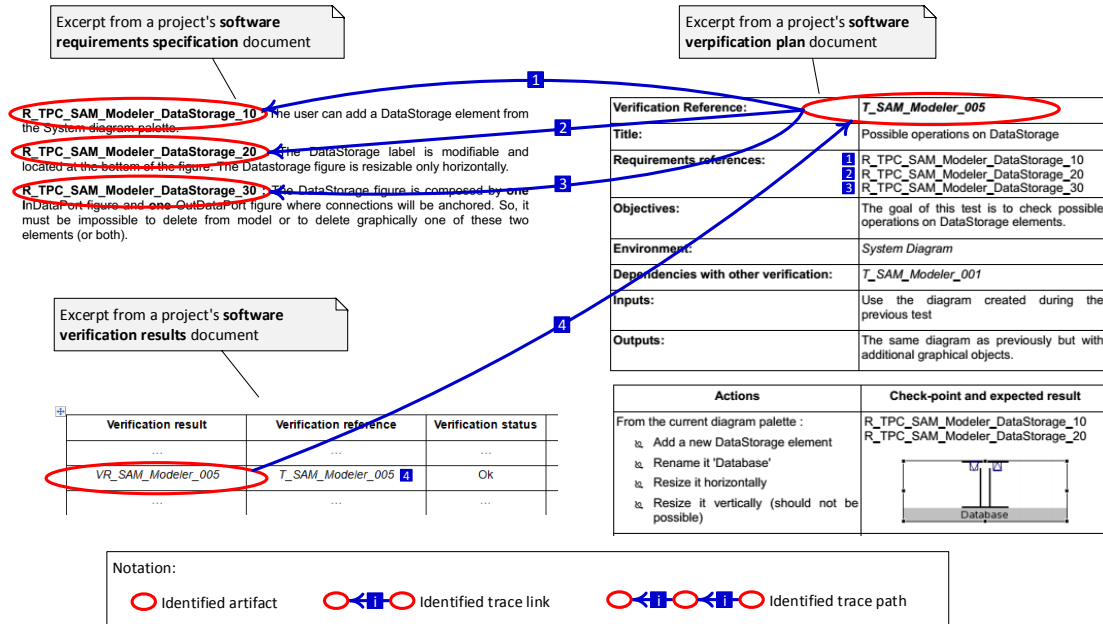


Figure 5.31.: Example of collecting artifacts, trace links, and trace paths from textual specification documents

As highlighted with red ellipses, the excerpt of the software requirements specification document contains three requirements artifacts: R-TPC-SAM-Modeler-DataStorage-10, R-TPC-SAM-Modeler-DataStorage-20, and R-TPC-SAM-Modeler-DataStorage-30. The excerpt of the software verification plan shows one verification procedure T-SAM-Modeler-005 that is supposed to verify the correct implementation of the three requirements artifacts. Accordingly, the verification plan document refers to the verified requirements via text references. These references are marked with blue shaded numbers in Figure 5.31. Each textual reference represents a trace link from the verification plan artifact to one of the three requirement artifact. Accordingly, these three trace links are illustrated with blue arrows in Figure 5.31. Since these textual references are merely created within the software verification plan, the trace links are only unidirectional and can only be traversed from the verification procedure to the referred requirements. This means, five artifacts (red ellipses) and four trace links (blue arrow) can be identified in the shown excerpt of the document. Additionally, the four trace links establish three unidirectional trace paths from the verification result artifact to the three requirements artifacts.

Figure 5.32 shows another example where artifacts and trace links are managed with the requirements management tool IBM Rational DOORS [IBM 2015b]. All artifacts and trace links are stored in a dedicated repository. A graphical user interface is provided to access the managed artifacts and trace links. It also provides the opportunity to automatically collect artifacts and trace links via the DOORS eXtension Language

5. Assessing the Fitness for Purpose of Implemented Traceability

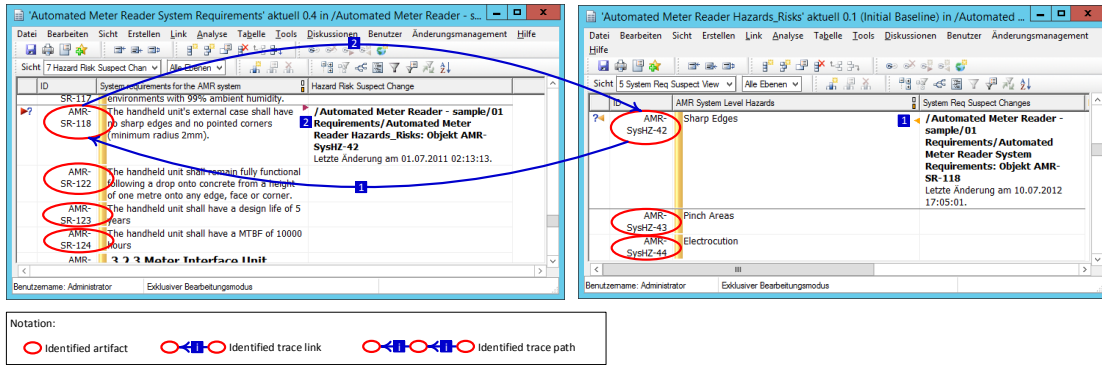


Figure 5.32.: Example of collecting artifacts, trace links, and trace paths from IBM Rational DOORS

(DXL). The left screen-shot in Figure 5.32 shows an excerpt of system requirements for an Automated Meter Reader, that are managed within a DOORS repository. As highlighted with red circles, four system requirement artifacts can be identified from the excerpt: AMR-SR-118, AMR-SR-122, AMR-SR-123, and AMR-SR-124. The right screen-shot in Figure 5.32 shows an excerpt of hazards and risks for an Automated Meter Reader, that are managed within the same DOORS repository. As highlighted with red circles, three system level hazards can be identified from the excerpt: AMR-SysHZ-42, AMR-SysHZ-43, and AMR-SysHZ-44. The two artifacts AMR-SysHZ-42 and AMR-SR-118 are connected by a bidirectional trace link, which can be traversed in both, forward and backward direction. This bidirectional trace link is illustrated as two blue arrows.

5.6.2. Step 2: Extracting Types from Implemented Traceability Data

Once all artifacts, trace links, and trace paths have been collected, artifact types, trace link types, and trace path types can be extracted from the collected data. Extracting these types is necessary to map the collected TID to the required traceability information, because these information are specified at a type level.

Depending on the representation of artifacts, the artifact type needs to be extracted from different sources. The simplest case is that an artifact has an explicit type property, which specifies the artifact type. If an artifact type is not specified explicitly, it needs to be extracted from the name and description of the context of an artifact. This context could be the section, module, document, or the like, in which the artifact is documented or stored.

Figure 5.33 illustrates the extraction of artifact types, trace link types, and trace path types from textual specification documents. Extractions from artifacts to artifact types and from trace links to trace link types are visualized as dotted lines. Extracted artifact types are shown as red rectangles. Extracted trace link types are displayed as blue arrows between the extracted artifact types.

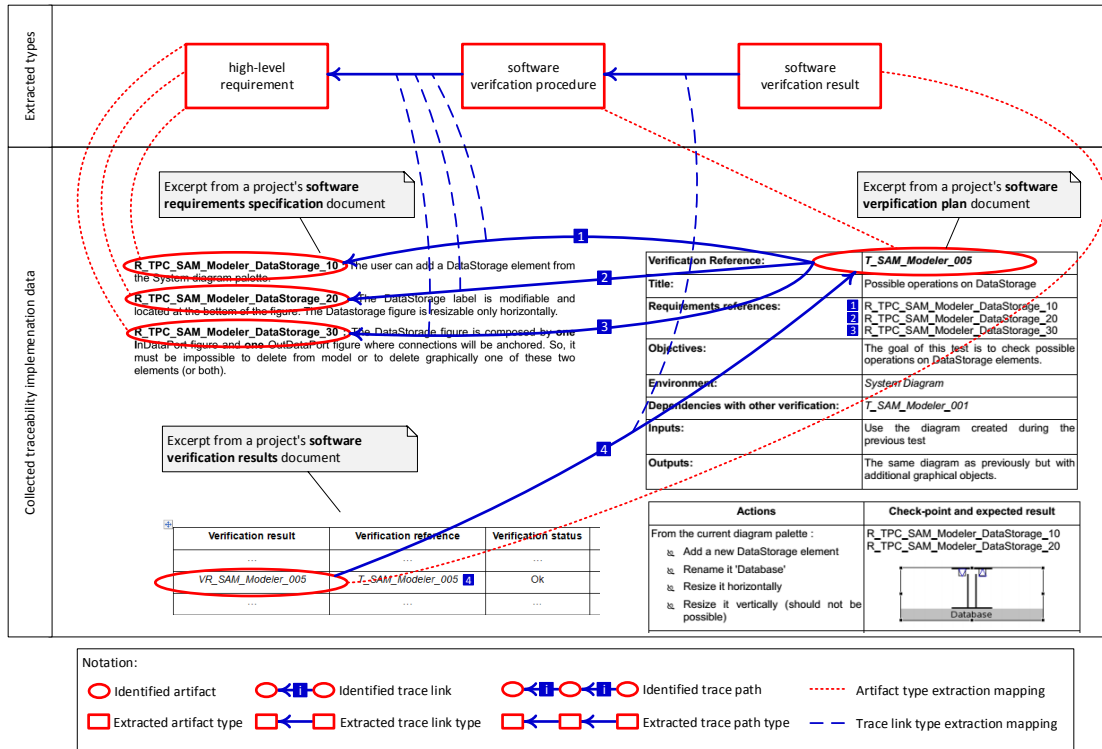


Figure 5.33.: Example of extracting artifact types, trace link types, and trace path types from textual specification documents

The artifacts R-TPC-SAM-Modeler-DataStorage-10, R-TPC-SAM-Modeler-DataStorage-20, and R-TPC-SAM-Modeler-DataStorage-30 belong to a section of the software requirements specification document, which is entitled with “Functional requirements”. Further, the “Purpose of the document” section states that the document focuses on specifying high-level software requirements. From this information, the artifact type *high-level requirement* can be derived for all three artifacts.

The artifact T-SAM-Modeler-005 belongs to a section of the verification plan document, which is entitled with “Description of the verifications”. From this information, the artifact type *software verification procedure* can be derived. Furthermore, the trace link type *high-level requirement* ← *software verification procedure* can be derived from the three trace links from T-SAM-Modeler-005 to R-TPC-SAM-Modeler-DataStorage-10, R-TPC-SAM-Modeler-DataStorage-20, and R-TPC-SAM-Modeler-DataStorage-30.

The artifact VR-SAM-Modeler-005 belongs to a section of the verification results document, which is entitled with “Verification results”. From this information, the artifact type *software verification result* can be derived. Furthermore, the trace link type *software verification result* ← *software verification procedure* can be derived from the trace link between VR-SAM-Modeler-005 and T-SAM-Modeler-005.

The extracted trace link types *high-level requirement* ← *software verification procedure*

5. Assessing the Fitness for Purpose of Implemented Traceability

and *software verification result* \leftarrow *software verification procedure* form a trace path type *high-level requirement* \leftarrow *software verification procedure* \leftarrow *software verification procedure*.

5.6.3. Step 3: Mapping Implemented Traceability Data to Required Traceability Information

Due to the extraction of types (see Section 5.6.2), TID are available at the same level of abstraction as the required traceability information within the TIM. To apply the proposed traceability assessment model to the extracted TID, the extracted artifact types need to be mapped to the corresponding required artifact types of the TIM. This mapping is the last necessary preparation step to apply the proposed traceability assessment model to an existing software development project.

Figure 5.34 exemplifies the mapping between the artifact types extracted from the project's traceability implementation data and the required artifact types of a TIM that was derived from the safety guideline DO-178B [DO-178B]. These required traceability information is relevant for the project [TOPCASE-SAM 2015], because it aims to comply with this guideline.

5.6.4. Step 4: Assessing the Implemented Traceability Data

After performing the three preparation steps, the resulting TID can be assessed for its fitness for purpose by applying the proposed TAM. As shown in figure 5.34, the following elements are available and can be used for performing an assessment: required artifact type, artifact type, artifact, required trace link type, trace link type, trace link, required trace path type, trace path type, and trace path. The corresponding required artifact types and artifact types are mapped.

In Section 5.4, all possible traceability problems were introduced and how the existence of this problem can be detected. Each of the formalized expressions can be applied to the resulting TID for detecting respective traceability problems. To illustrate the application of the formalized expression, an example is provided for the traceability problem missing artifact type ($\mathcal{M}_{\mathcal{A}}$), which is related to incomplete traceability. For demonstration purposes, it is assumed that the shown excerpt in Figure 5.34 represent the complete set of TID for the project.

The following facts are shown in Figure 5.34. The TIM specifies the following required artifact types: $\mathcal{R}_{\mathcal{A}} = \{\text{SyR, HLR, SwA, LLR, SC, TC, TP, TR}\}$. The project implements the following artifact types: $I_{\mathcal{A}} = \{\text{Impl-HLR, Impl-SVP, Impl-SVR}\}$. The artifact type Impl-HLR implements the required artifact type HLR so that: $implements(\text{Impl-HLR}) = \{\text{HLR}\}$. The artifact type Impl-SVP implements the required artifact types TC and TR so that: $implements(\text{Impl-SVP}) = \{\text{TC, TR}\}$. The artifact type Impl-SVR implements the required artifact types TC and TP so that: $implements(\text{Impl-SVR}) = \{\text{TR}\}$. The formal expression

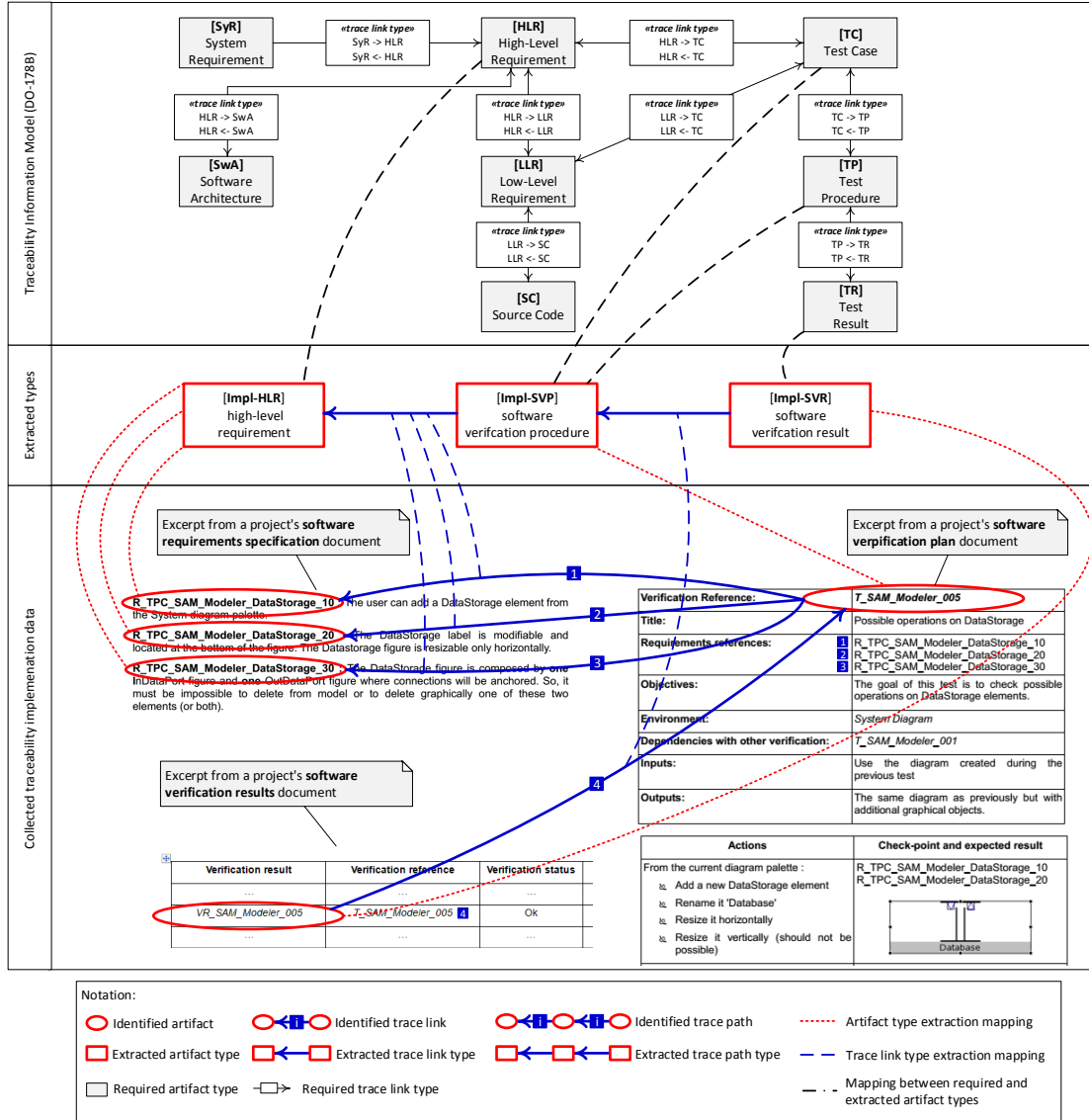


Figure 5.34.: Example of mapping the extracted artifact types with the required artifact types of the TIM

for identifying missing artifact types is defined as: $\exists a \in \mathcal{R}_{\mathcal{A}}[\text{implements}^{-1}(a) = \emptyset] \implies a \in \mathcal{M}_{\mathcal{A}}$. As specified by the formal expression, for each element that is member of the set $\mathcal{R}_{\mathcal{A}}$ (all required artifact types) it is necessary to assess if it is implemented by a corresponding artifact type. In the following, the evaluations of the formal expression are shown for each required artifact type:

- $(\text{implements}^{-1}(\text{SyR}) = \emptyset)$ is *TRUE*.
- $(\text{implements}^{-1}(\text{HLR}) = \emptyset)$ is *FALSE*.

5. Assessing the Fitness for Purpose of Implemented Traceability

- $(implements^{-1}(SwA) = \emptyset)$ is *TRUE*.
- $(implements^{-1}(LLR) = \emptyset)$ is *TRUE*.
- $(implements^{-1}(SC) = \emptyset)$ is *TRUE*.
- $(implements^{-1}(TC) = \emptyset)$ is *FALSE*.
- $(implements^{-1}(TP) = \emptyset)$ is *FALSE*.
- $(implements^{-1}(TR) = \emptyset)$ is *FALSE*.

For the example shown in Figure 5.34 it is implied that the set of missing artifact types contains four elements: $\mathcal{M}_{\mathcal{A}} = \{SyR, SwA, LLR, SC\}$. This would lead to the assessment result that four required artifact types are missing. Each formal expression, which is provided in Section 5.4 can be used in the same way as illustrated for the traceability problem missing artifact type to quantify the number of existing traceability problems within a project specific traceability implementation. These quantified traceability problems can be summarized in a traceability assessment report.

Element <i>Instance of</i>	Implementation data (available elements)	Completeness problems (missing elements)		Appropriateness problems (superfluous elements)		Correctness problems (wrong elements)	
	absolute	absolute	relative	absolute	relative	absolute	relative
artifact type	7	2	28,57%	1	14,29%	-	-
trace link type	10	3	30,00%	1	10,00%	0	0,00%
trace path type	6	4	66,67%	0	0,00%	0	0,00%
artifact							
<i>All</i>	588	-	-	17	2,89%	-	-
<i>HLR</i>	102	-	-	0	0,00%	-	-
<i>SVC</i>	67	-	-	0	0,00%	-	-
trace link							
<i>All</i>	1013	71	7,01%	32	3,16%	0	0,00%
<i>SVC -> HLR</i>	71	8	11,27%	14	19,72%	0	0,00%
trace path							
<i>All</i>	495	32	6,46%	11	2,22%	0	0,00%
<i>VR -> SVC -> HLR</i>	57	5	8,77%	0	0,00%	0	0,00%

Figure 5.35.: Traceability assessment report example

Figure 5.35 shows a traceability assessment report example, which summarizes the number of problems per element type and problem type. The first column enumerates the assessed TID element names. The second column outlines the numbers' frequency of the TID. The third column summarizes absolute and relative frequencies of detected completeness problems. The fourth column outlines absolute and relative frequencies of detected appropriateness problems. The last column summarizes absolute and relative frequencies of detected correctness problems.

Apart from quantifying the traceability problems of an assessed project, the problem definitions provided in Section 5.4 can also be used to provide detailed reporting about the location of the problems. Problems related to appropriateness and correctness of traceability are the simplest cases for detailed reporting, because the entity itself exists, which causes the traceability problem. Accordingly, the concrete problem causing traceability entity can be reported. However, reporting problems of incompleteness is more challenging, as the traceability entity itself is missing and coherently does not exist. Missing artifact types and missing artifacts are caused by a required artifact type within the TIM. Accordingly, the name of the required artifact type from the TIM can be reported. Missing trace link types and missing trace path types are caused by a required trace link type or a required trace path type of the TIM, whose name can be reported as well. A missing trace link and a missing trace path always corresponds to an existing artifact for which the trace link or trace path is missing. Accordingly, the name of the artifact that misses the trace link or trace path can be reported.

5.7. Summary

This chapter has presented a method for assessing the fitness for purpose of a project's traceability implementation. It represents the second and core part of the proposed traceability assessment approach (see Section 3.3). Main objective of this part is to ensure trusted traceability. In this section, a summary of the presented method is provided with respect to the challenges that are related to trusted traceability (see Section 3.1.2).

As stated in Challenge 3, the term "fitness for purpose" needs to be defined clearly by the traceability assessment approach. To address this challenge, the relevant quality attributes completeness, appropriateness, and correctness were identified and defined with respect to the traceability assessment context (see Section 5.2).

The demand for clear assessment criteria is expressed in Challenge 4. This challenge is addressed by providing formal expressions for each assessment criterion. Illustrating examples are provided for all assessment criteria (see Section 5.4). Additionally, the application of one assessment criterion is illustrated Section 5.6.

Challenge 5 emphasized that the assessment approach needs cover the detection of all relevant traceability problems. To ensure a complete coverage, assessment criteria were defined for all elements of the TID and all three quality attributes. Additionally, this aspect was qualitatively evaluated with traceability experts. The results of this evaluation are presented in Section 7.3.

Assessing a project's traceability implementation for its fitness for purpose is the main part of the proposed traceability assessment approach (see Section 3.3). The following chapter will discuss the implementation of prototype that automates the assessment approach.

6. Tool Support for Continuous Traceability Assessment

This chapter discusses the implementation of a prototype called PurISTA. The PurISTA tool was implemented to automate the assessment of a project's traceability implementation for its fitness for purpose.

Section 6.1 provides an overview of the PurISTA tool, which consists of five components. Each component is discussed with respect to relevant implementation details. Section 6.2 summarizes the prototypical implementation of the traceability assessment approach with respect to the identified challenges that are related to automation (see Section 3.1.3).

6.1. The PurISTA Prototype

The PurISTA tool was implemented using the Microsoft .Net Framework 4.5 platform. As depicted in Figure 6.1, the PurISTA tool consists of the following five components:

- The *traceability store* is the central data store that contains all traceability planning and implementation data (see Section 6.1.1).
- All the project specific traceability planning data are established through the *traceability planner* component (see Section 6.1.2) and stored in the traceability store.
- All the project specific traceability implementation data are collected via the *traceability collector* component (see Section 6.1.3) and stored in the traceability store.
- The *traceability browser* can be used to visualize collected traceability planning and implementation data to browse through the traceability network (see Section 6.1.4).
- The *traceability assessor* makes use of the traceability planning and implementation data from the traceability store to assess the fitness for purpose of the actual traceability implementation (see Section 6.1.5).

The PurISTA tool features a service-oriented architecture [Huhns and Singh 2005]. Each component is a self-contained unit of functionality.

6. Tool Support for Continuous Traceability Assessment

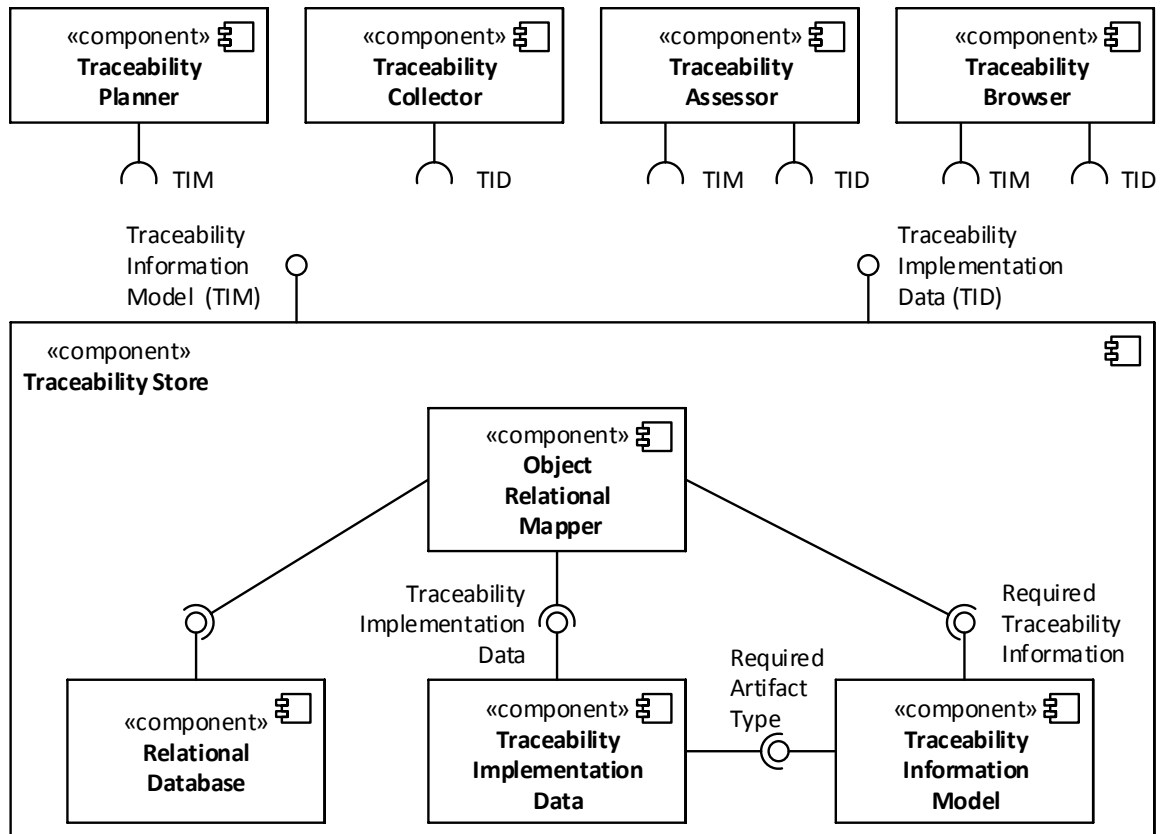


Figure 6.1.: Overview of the PurISTA tool components

6.1.1. Traceability Store

The traceability store is the component that persists all traceability related data, which are necessary to automate the assessment approach as introduced in Section 5. The following two design goals were formulated when designing the persistence layer. First, the persistence layer should provide an object-oriented access to the persisted data. The reasoning for this design goal is that the traceability planning and assessment approach (see also Chapter 4 and 5) rests on object-oriented models such as TIM, TRM, and TID. Second, the persistence layer should not be dependent on a specific Relational Database Management System (RDBMS) technology. This design goal should guarantee a more general applicability of the PurISTA tool, which is not restricted by severe RDBMS technology constraints. Since the PurISTA tool is developed on the Microsoft .Net Framework 4.5 platform, the object relational mapping platform NHibernate [NHibernate 2015] was chosen to develop the persistence layer. It runs on the Microsoft .Net Platform and supports numerous relational databases, such as MS SQL Server [Microsoft 2015a], Oracle Database [Oracle 2015c], IBM DB2 [IBM 2015a], MySQL [Oracle 2015b], and SQLite [SQLite 2015].

To provide an object oriented access to the persistence layer, two models were created. One model for traceability requirements data and another model for traceability implementation data. Each entity that belongs to a model is defined as follows: First, a class is created that represents entity. Thereby, the properties of an entity are represented by class properties. Second, for each entity a NHibernate mapping file is created, which defines how an entity is persisted within the relational data store. Figure 6.2 shows an example of a model entity to relational database mapping definition. The mapping is defined with eXtensible Markup Language (XML) files that must conform to an XML schema, which was defined by the NHibernate project.

```

<?xml version="1.0" encoding="utf-8" ?>
<hibernate-mapping xmlns="urn:hibernate-mapping-2.2"
  assembly="TraceabilityConformance" ❶
  namespace="TraceabilityConformance.Model">
  <class name="Artifact" table="Artifacts">❷
    <!-- IdentifiableElement -->
    ❸ <id name="Id" generator="increment"/>
    <property name="Uid" not-null="true" />
    ❹ <property name="Anchor" not-null="true"/>
    <!-- NamedElement -->
    <property name="Name" />
    <!-- Artifact -->
    ❺ <many-to-one name="Type" column="ArtifactTypeId" not-found="exception" not-null="true"/>
    <many-to-one name="Context" column="ContextId" not-found="exception" not-null="true"/>
  </class>
</hibernate-mapping>

```

Figure 6.2.: Example of a model entity to relational database mapping definition

As highlighted with ❶, the element `hibernate-mapping`, defines the mapping for one entity. The attribute `assembly` and `namespace` specify the class that implements the entity. The `class` element (❷) defines the class name of the entity with the attribute `name` and relational table name where the entity is stored with the attribute `table`. The `id` element (❸) specifies the column name that represents the primary key and how the primary key values are created for new entities. The `property` elements (❹) specify entity properties of primitive types. The `name` attribute specifies the property name of the entity class and the column name of the relational table that represents the entity. Relations to other entities or non primitive types are defined by `one-to-one`, `many-to-one`, and `many-to-many` elements. The first `many-to-one` element (❺) defines an entity property named `Type` which contains values that are represented by instances of another entity. Accordingly, a foreign-key column is defined by the attribute `column`. Another technical configuration file defines the connection parameter to the relational data store that persists model entity instances.

6. Tool Support for Continuous Traceability Assessment

Once, all entity classes are implemented and all entity mapping files are created, the NHibernate framework can be used to initially generate the relational database schema and then to create, read, update, delete, and query entities in an object oriented manner. As depicted in Figure 6.1, the traceability store component provides two public data access interfaces. One interface provides access to the entities of the traceability implementation model. The other interface provides access to the entities of the traceability requirements model. Therefore, the access to the different models can be managed and controlled independently.

6.1.2. Traceability Planner

A main prerequisite for assessing the fitness for purpose of a project's traceability implementation is to plan for purposed traceability as discussed in Section 4. The traceability planner component can be used to specify and import the traceability requirements, which were derived by the creation of the TIM and the Traceability Goal Model (TGM). For the specification of the traceability requirements, predefined spreadsheet templates are provided, which are filled by the person who is responsible for planning the traceability. The traceability planner can automatically import and parse the data from the filled spreadsheet. The traceability planner component communicates with the traceability requirements model interface of the traceability store component to persist the collected traceability requirements data.

6.1.3. Traceability Collector

The main purpose of the traceability collector component is to collect and parse traceability implementation data of a software development project. These project implementation data are typically not managed in one homogeneous tool. Instead, different special purpose tools are used depending on the focus of an artifact. For example, requirement artifacts are typically managed by specialized tools such as Rational DOORS [IBM 2015b] or general purpose word processors such as Microsoft Word [Microsoft 2015b] or Open Office [Apache 2015a]. Source code artifacts are mostly managed by Version Control Systems (VCS) such as, for example, Git [Git Team 2015], Subversion [Apache 2015b], or Mercurial [Mercurial Team 2015]. Due to the fact that traceability implementation data are managed by different tools, the traceability collector needs to be capable of collecting these data from multiple sources.

As depicted in Figure 6.3, the traceability collector implements interfaces to different external artifact management tools to automatically collect traceability implementation data. Rational DOORS [IBM 2015b] is typically used to manage artifacts such as system requirements, software requirements, low-level requirements, risks, hazards, and test cases. Trace links are stored as independent artifacts and can be traversed in both, forward and backward direction. The tool provides a programmable interface

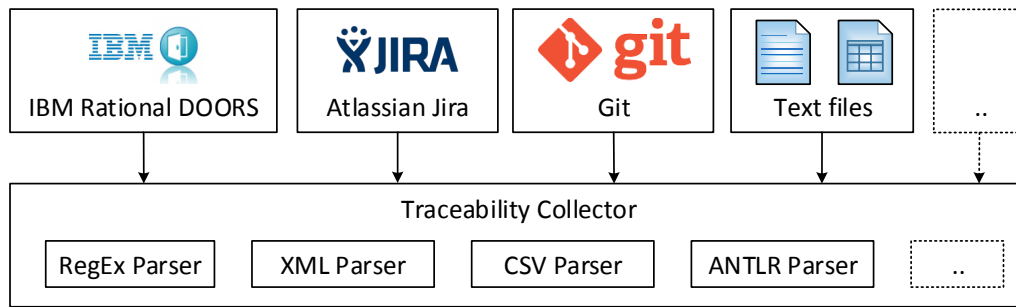


Figure 6.3.: Overview of implemented external interfaces with artifact management tools

DXL through which each artifact and trace link can be retrieved as an atomic element. Atlassian Jira [Jira 2015] is the second external tool from which artifacts can automatically be collected. The tool, a widely spread issue tracking tool for managing artifacts such as features, bugs, or tasks. All managed artifacts and trace links are accessible through a web service. Since the artifacts and trace links are provided as XML structures, an additional XML parser was implemented to extract the relevant traceability implementation information. As source code artifacts are typically managed by VCS, the traceability collector implements an interface to Git [Git Team 2015], which is a prominent representative of this kind of tools. A git repository can be remotely accessed via the git access protocol. The prototype leverages the open source .Net LibGit2Sharp [LibGit2Sharp Team 2015] for implementing this access protocol. To extract artifact properties from the versioned source code files such as, for example, identifier and package name of each implemented class, a parser for the Java programming language [Oracle 2015a] was implemented by using the parser generator framework ANTLR [ANTLR Team 2015a]. The parser was generated with the “Java 1.7 grammar for ANTLR v4”, which is provided in [ANTLR Team 2015b]. Trace links within Git are typically represented within the commit messages as textual references. For this purpose, a regular expression parser was implemented which searches for artifact identifier references that conform to a specific pattern. Due to the fact that many artifacts are managed with general purpose word processors, the traceability collector also implements an interface to import text documents. Depending on the provided format, different text parsers (XML, comma separated value, or regular expressions) are used to extract the containing artifacts and textual references. As indicated with dotted boxes in Figure 6.3 additional interfaces to artifact management tools as well as additional parsers can be implemented if needed. Since the collector can never provide a full coverage of all available artifact management tools, this data integration is always a project specific integration effort. In the future, the currently developed Open Services for Lifecycle Collaboration (OSLC) stan-

dard could be a feasible solution to sufficiently address this data integration problem, provided that the standard achieves acceptance by the tool vendors. The traceability collector component communicates with the traceability implementation model interface of the traceability store component to persist the collected traceability implementation data.

6.1.4. Traceability Browser

The main purpose of the traceability browser component is to provide a graphical user interface for the traceability engineer to review the planned and the implemented traceability data, which were established by the traceability planner and traceability collector components. The Windows Presentation Foundation (WPF) was leveraged to implement the graphical user interface. The component features the Model View ViewModel (MVVM) design pattern [Smith 2009].

Figure 6.4 shows a screenshot of the traceability browser component at the lowest zoom level zero. As marked with ❶, the green framed box in the center of the diagram visualizes the traceability requirements that were derived from the safety standard DO-178B [DO-178B]. The other two green framed boxes, which are marked with ❷, represent traceability implementation data at the artifact type level of two different development projects. As annotated with ❸, the green dotted lines represent mappings of the projects' artifact types to the corresponding required artifact types of the DO-178B standard.

Since these traceability networks can get very comprehensive, the user can interactively zoom into and browse through the visualized structure. Further, the visualization of certain edges within the graph can be disabled. The Figure 6.5 visualizes the same traceability requirements as an implementation data at zoom level one. The visualization of all edges except of required trace link types and required trace path types are disabled. The context of the visualized artifact types is still visualized by a green frame (❶). The selected vertex is arranged in the center of the diagram (❷). Starting from this selected vertex, only the related vertices (❸) are shown which are directly related through any edge (❹). At this zoom level, only the selected vertex and all directly related vertices in the graph are shown. By clicking on one of the connected vertices, this vertex becomes selected and for this vertex all related vertices are shown. Thereby, the user of the tool can browse through the entire graph of connected artifact types.

Since zoom level zero and one only visualize vertices at the granularity level artifact types, the user can further zoom into the visualized structure. Figure 6.6 shows a screenshot of traceability browser that visualizes the traceability implementation data at artifact level, which corresponds to zoom-level two. While the artifact types are represented as green frames (❶), the corresponding artifact instances are shown as vertices (❷) within the frame. Trace links between artifacts are represented as dotted black lines (❸). The user can further zoom into the model to only show the selected artifact and

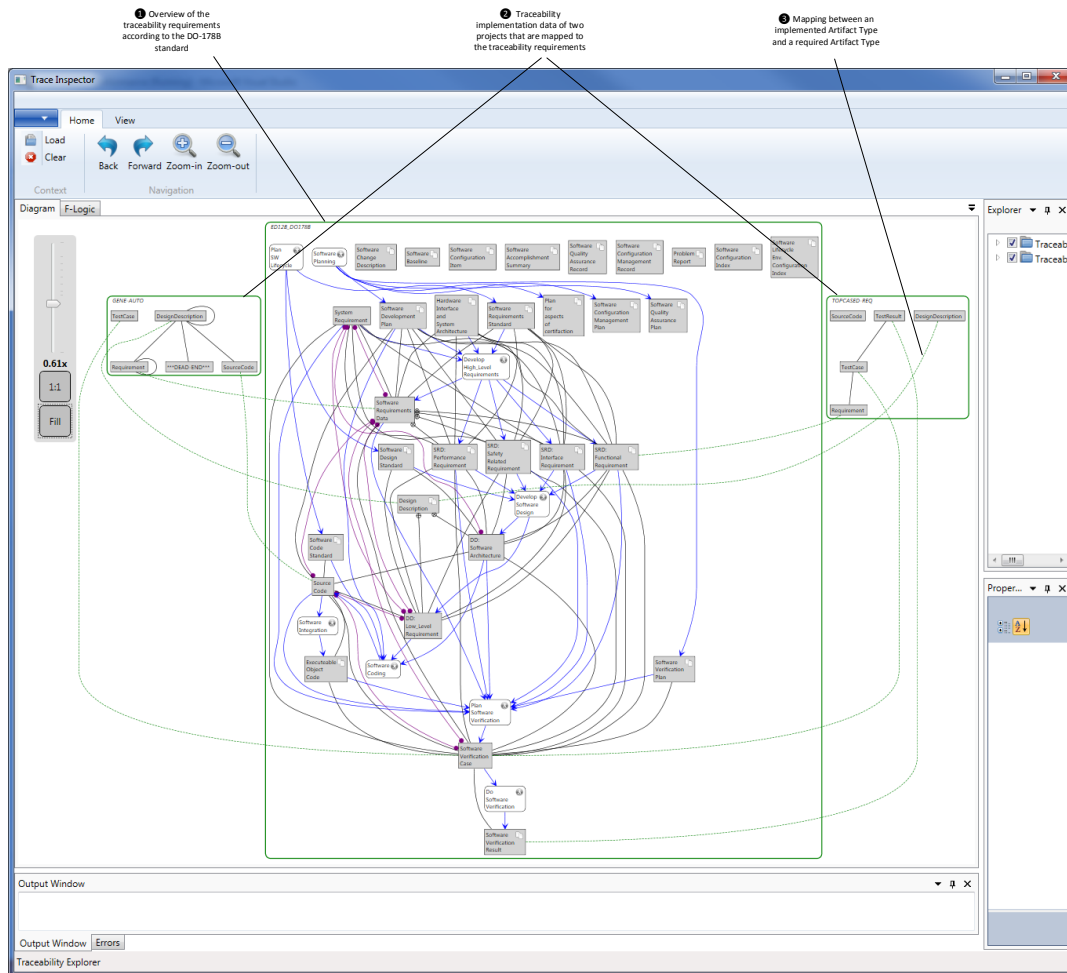


Figure 6.4.: Screenshot of the traceability browser that visualizes the traceability requirements derived from the DO-178B standard [DO-178B] and the artifact types of two projects at zoom level zero

all directly related artifacts. Similarly to zoom level one, the user can browse through all the connected artifacts by clicking on a directly related artifact, which becomes interactively the selected vertex.

6.1.5. Traceability Assessor

Once, the traceability requirements data and the traceability implementation data are imported to the traceability store, the traceability assessor can analyze the persisted traceability implementation data for its fitness for purpose.

To enable an effective analysis, an in-memory graph structure is created from the persisted data. For this purpose, the traceability assessor component leverages the open

6. Tool Support for Continuous Traceability Assessment

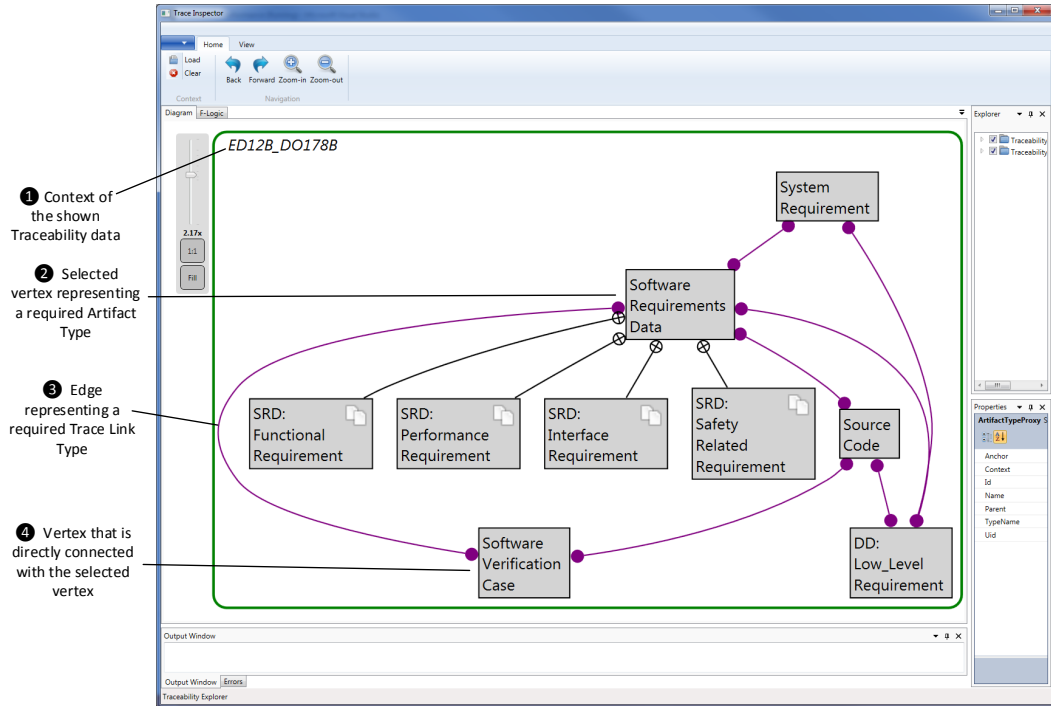


Figure 6.5.: Screenshot of the traceability browser shows the currently selected artifact type and all directly related elements

source library QickGraph [Jonathan de Halleux 2015]. This library provides ready to use implementation of graph algorithms to solve standard graph problems. For example, the implementation of the Dijkstra shortest-path algorithm [Dijkstra 1959] is used by the traceability assessor to derive for any vertex in the graph the sub-graph of all connected vertices. The traceability assessor component analyzes the in-memory graph representation for potential traceability problems. Therefore, it iterates over all vertices of the graph. For each vertex, it checks the existence of traceability problems as presented in Section 5.4. For each problem, one problem analysis rule is implemented. These rules are applied for each assessed vertex of the in-memory graph sequentially.

Figure 6.7 shows an example assessment report of identified traceability problems, which was produced by the traceability assessor. The area marked with ❶ reports identified traceability problems with respect to missing trace path types (\mathcal{M}_p). The report area marked with ❷ lists all required trace link types and required trace link paths. The identified traceability problems are listed in the report area marked with ❸. The area marked with ❹ shows the quantity of identified problems. The quantities of identified problems can also be exported to a spreadsheet. As a result, similar to the manual assessment traceability assessment as exemplified in Section 5.6.4, the same result spreadsheet as depicted in Figure 5.35 is generated by the traceability assessor.

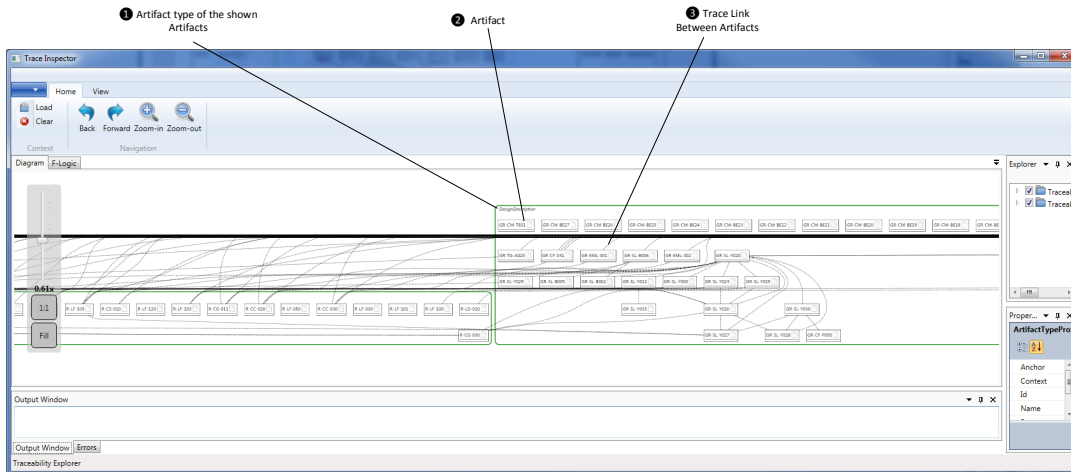


Figure 6.6.: Screenshot of the traceability browser that visualizes the traceability implementation data at artifact level

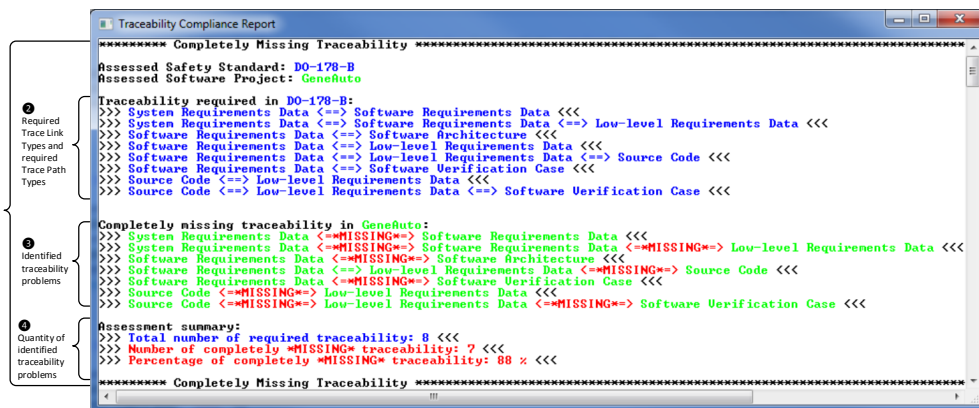


Figure 6.7.: Example of an assessment report for the traceability problems missing trace path type (\mathcal{M}_p)

6.2. Summary

This chapter has presented a prototypical implementation of the proposed traceability assessment approach. Main objective of the PurISTA tool is the automation of the traceability assessment approach. This section provides a summarizing discussion of the implemented prototype with respect to the challenge that is related to the automation (see Section 3.1.3).

As stated in Challenge 6, the automated assessment of a project's entire traceability implementation should be supported. To support the assessment of a project's entire traceability implementation, the traceability collector implements interfaces to a variety

6. *Tool Support for Continuous Traceability Assessment*

of external artifact management tools. These interfaces were sufficient to automatically assess any traceability implementation data of four software development projects that were aiming to comply with safety related guidelines (see Section 7.4). However, since the collector can never provide a full coverage of all available artifact management tools, this data integration is always a project specific integration effort. Although the initial TID import potentially requires project specific adaptations to cover all artifact management tools, the assessment of the imported data is fully automated by the traceability assessor component, which does not require any project specific adoptions. It executes in-memory traceability graph assessment rules. Each rule corresponds to the one of the formal expressions that are provided in Section 5.4.

7. Evaluation

This chapter outlines the evaluation of the proposed traceability assessment approach. In preparation for evaluating the approach, seven research questions were derived from the traceability assessment challenges (see Section 3.1) and the usage scenarios of traceability assessment (see Section 3.1). Four studies have been undertaken to answer these research questions. The first study collected feedback from 12 traceability experts on the TAM that was introduced in Section 5.1. The second study conducted interviews with stakeholders from 17 software projects to evaluate the traceability assessment approach (see Section 3.3) with respect to the traceability driver *value*. In a third study, the traceability assessment approach was evaluated with respect to the traceability driver *regulation*. Therefore, the approach was applied to the TID of four safety-critical software projects. The fourth study collected feedback from 17 safety project participants and certifiers.

In Section 7.1, the research questions will be derived. Section 7.2 refers to the first study that focuses on the evaluation of the TAM. The second study is presented in Section 7.3, evaluating the assessment approach within the context of projects that are driven by the value of traceability. Section 7.4 refers to the third study, evaluating the approach for domains that are regulated by safety guidelines. In Section 7.5, the questionnaire survey with 17 safety project participants and certifiers is outlined. Section 7.6 provides a discussion of the study results with respect to the research questions. In Section 7.7, potential threats to validity will be discussed, as well as how these threats have been mitigated.

7.1. Research Questions

This section defines research question for the evaluation of the proposed traceability assessment approach. These questions are derived from identified traceability assessment challenges (see Section 3.1) and from intended usage scenarios of the assessment approach (see Section 3.4).

To establish trust in the assessment results, the traceability assessment approach needs to ensure that all traceability problems can be detected with respect to the fitness for purpose (see Challenge 5). This implies that the assessment criteria of the TAM need to cover *all* traceability problems and each traceability problem should be relevant with respect to the fitness for purpose. Hence, the following two research questions can

7. Evaluation

be derived:

Research Question 1 (Relevance). *Does the problem classification of the defined TAM specify relevant traceability problems with respect to the fitness for purpose of a project's traceability implementation?*

Research Question 2 (Completeness). *Does the problem classification of the defined TAM completely cover all relevant traceability problems with respect to the fitness for purpose of a project's traceability implementation?*

Traceability is required by numerous software lifecycle activities such as, for example, safety analysis, change impact analysis, coverage analysis, and compliance verification. This requires the planning for purposed traceability (see Challenge 1), which supports the planning for multiple purposes (see Challenge 2). The set of trace links required by one activity can be different to the set required by another activity. Thus, the feasibility of a particular activity depends on the completeness and correctness of the respective set of trace links. As claimed in Scenario 5, the assessment approach provides support for determining the feasibility of software lifecycle activities. Thus, the following research question can be derived:

Research Question 3 (Feasibility of software lifecycle activities). *Can the assessment approach be used to determine the feasibility of a software lifecycle activity that requires traceability?*

The manual creation of trace links is cost-intensive [Cleland-Huang et al. 2004; Heindl and Biffel 2005]. Hence, the creation of superfluous trace links should be avoided, in order to ensure a cost effective traceability implementation. This requires the planning for purposed traceability (see Challenge 1), which supports the planning for multiple purposes (see Challenge 2). The description of Scenario 6 envisions the support for determining the cost-effectiveness of a traceability implementation, which leads to the following research question:

Research Question 4 (Cost-effective implementation). *Can the assessment approach be used to determine if a project's traceability implementation is cost-effective?*

Safety critical software products need to be certified by the responsible authority before they can be released to the market. During this certification process, the authority checks whether or not the developed systems can be considered as safe to be used. As part of this certification process, certifiers check compliance of the implemented traceability with traceability requirements of the relevant guideline. A product may need to comply to multiple guidelines if it is released into multiple markets. The Scenarios 1 and 3 envision that the assessment approach provide support for these compliance checks:

Research Question 5 (Compliance). *Can the assessment approach be used to determine if a project's traceability implementation complies to one or many guidelines?*

When an existing product is introduced into a new market, it may be necessary to certify the product under a new guideline. Similarly, existing guidelines may be revised (for example, DO-178B → DO-178C) and the new version becomes immediately relevant for product development. As claimed in Scenario 4, these guideline migrations can be supported:

Research Question 6 (Migration). *Can the assessment approach be used to support the migration to a new guideline?*

Consistently maintaining a project in a ready-to-certify state requires a rigorous assessment process built into the development environment, continuous integration, and accurately maintained traceability, available at any time to support the certification process. The support of a continuous assessment is claimed in Scenario 2, leading to the following research question:

Research Question 7 (Continuous assessment). *Can the assessment approach be used to assess a project's traceability implementation in a continuous manner?*

These research questions were used for the evaluation of the proposed traceability assessment approach.

7.2. Study 1: Traceability Assessment Model

This section describes a survey that was carried out with traceability experts, in order to evaluate the proposed TAM (see Section 5.1). This study was partly published in [Rempel and Mäder 2015a].

Study objectives. In this study, the TAM was evaluated with respect to the Research Question 1 and 2 (see Section 7.1). The TAM provides criteria that can be used to detect traceability problems (see Section 5.4). Each problem specifies a situation where an element of the TID is either missing, superfluous, or wrong with respect to the required traceability information as defined within the TIM. Two study objectives can be derived from the research questions. To answer Research Question 1, the relevance of each problem type needs to be evaluated. The completeness of the entire problem type classification needs to be evaluated to answer Research Question 2.

Study instrument. Experienced traceability experts with extensive industrial experience are a good source to provide reliable judgments. Surveys provide effective means to generalize about opinions of many experts by studying a subset of them [Kitchenham and S. L. Pfleeger 2008]. The above stated study objectives demand for a descriptive survey,

7. Evaluation

as described in [Wohlin et al. 2012]. The survey for this study was conducted through questionnaire method, which can define standardized opinion scales. This method allows to produce quantitative survey results. Based on these results, averages and trends can be determined with respect to the experts' opinions.

Target audience of the questionnaire survey. Evaluating the relevance of each problem and the completeness of the problem classification requires practical expertise with software traceability. To characterize the population of traceability experts, the international requirements engineering conference was used as reference [RE 2015], because traceability is an explicit topic of interest that was introduced at the first issue of this conference [Gotel and C. Finkelstein 1994]. Between 1994 and 2013, traceability contributions were mainly coming from academia (70%), allocated to the following institutions: the University of Kentucky (9), University of Toronto (7), DePaul University (6), Johannes Kepler University (4), and City University London (4) [Nair et al. 2013]. These numbers indicate that the population of traceability experts is relatively small. Authors of traceability related contributions to this conference were considered as target audience for the survey. This target audience was extended by authors who contributed to the international symposium on software and systems traceability [SST 2015].

Sampling strategy. The institutions were ranked based on the number of contributions. To avoid bias from accidental sampling, the quota sampling approach was applied [Thompson 2012]. The participants of the questionnaire survey were limited to two participants per institution.

Data collection. The feedback from traceability experts was collected via a questionnaire, which comprised five parts: 1) introduction, 2) preliminary questions, 3) software traceability definitions, 4) the traceability problem classification, and 5) a debriefing. In the introduction, the purpose of the assessment approach was introduced. This part also introduced the scales that were used throughout the questionnaire to ask each subject for his or her rating of each traceability problem classification. The second part asked questions about the traceability expertise of each subject. In the third part, illustrating diagrams were provided to make the subject familiar with the preliminary assumptions from which the traceability problems were derived. In the fourth part, each subject was requested to rate the level of importance for each problem. Therefore, an ordinal scale was used ranging from 1 (minor) to 5 (major). In the last part, each subject was asked to state any structural problem that was missing in the classification. Any subject's response was considered as agreement to the completeness of the classification, if no missing problem type was mentioned. The complete questionnaire is provided in Appendix A.

Over a period of two weeks, 13 of the 22 contacted subjects returned a completely filled

7.3. Study 2: Value Driven Traceability Implementations

questionnaire. The participating subjects had an average practical software traceability experience of 8.61 years. Additionally, 12 out of the 13 participants had either worked in a project that captured traceability information or had professionally assessed the quality of captured traceability information in a software project. Ten participants answered that they had done both.

Data Analysis. The returned questionnaires data were analyzed to quantify the traceability experts opinion on the relevance per traceability problem and the completeness of the traceability problem classification. The subjects were requested to rate each traceability problem's relevance on an ordinal scale between 1 (minor) and 5 (major). Based on these opinion scores, aggregated values over all subjects were calculated. The last part asked the subjects to enumerate the problem types that are missing in their opinion. Depending on the answer provided by each subject to this question, the completeness was either coded as *true* (no missing problem type were enumerated) or *false* (one or many missing problem types were enumerated).

Results. Figure 7.1 compares the relevance ratings given by the 13 subjects for the traceability problems introduced in Section 5.4. The individual statistics for each traceability problem type are as follows: \mathcal{M}_A - missing artifact type (avg: 4.61, med: 5, std: 0.87), \mathcal{M}_L - missing trace link type (avg: 4.38, med: 5, std: 0.77), \mathcal{M}_P - missing trace path type (avg: 4.15, med: 4, std: 0.69), \mathcal{M}_A - missing artifact (avg: 3.23, med: 3, std: 1.24), \mathcal{M}_L - missing trace link (avg: 3.69, med: 4, std: 1.03), \mathcal{M}_P - missing trace path (avg: 3.46, med: 4, std: 1.05), \mathcal{S}_L - superfluous trace link type (avg: 2.15, med: 2, std: 0.99), \mathcal{S}_P - superfluous trace path type (avg: 2.38, med: 3, std: 0.77), \mathcal{S}_L - superfluous trace link (avg: 3.15, med: 3, std: 1.28), \mathcal{S}_P - superfluous trace path (avg: 3.07, med: 3, std: 1.32), \mathcal{W}_L - wrong trace link type (avg: 4, med: 4, std: 0.71), \mathcal{W}_P - wrong trace path type (avg: 3.92, med: 4, std: 0.64), \mathcal{W}_L - wrong trace link (avg: 3.69, med: 4, std: 1.03), \mathcal{W}_P - wrong trace path (avg: 3.46, med: 4, std: 0.97).

The traceability problem classification was rated by 12 subjects as complete. 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”.

7.3. Study 2: Value Driven Traceability Implementations

This section outlines a survey that was carried out with software engineers from industrial software development projects to evaluate the assessment approach for value driven traceability implementations. This study was partly published in [Rempel et al. 2013].

Study objectives. In this study, the proposed traceability assessment approach was

7. Evaluation

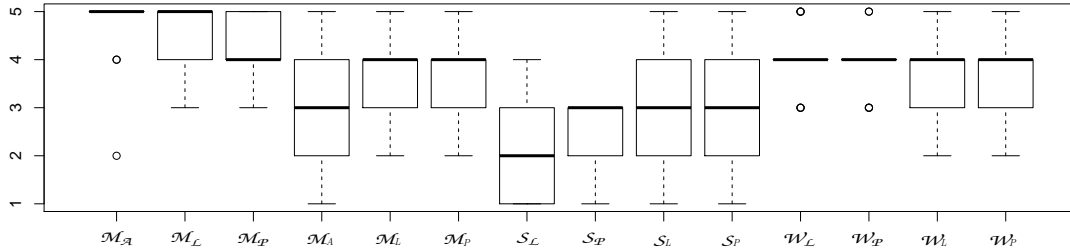


Figure 7.1.: Results of study 1: average relevance ratings per traceability problem

evaluated with respect to the Research Question 3 and 4 (see Section 7.1).

Two study objectives can be derived from the two research questions. To answer Research Question 3, the assessment approach needs to be evaluated for its capability to determine whether or not a software lifecycle activity that requires traceability is feasible. To answer Research Question 4, the assessment approach needs to be evaluated for its capability to determine whether or not an existing traceability implementation is cost-effective.

Study instrument. Stakeholder specific traceability goals need to be identified, in order to plan for purposed traceability (see Section 4.1.2). Therefore, relevant stakeholders of the project, that may be concerned with the result of a software lifecycle activity requiring traceability, need to be interviewed to identify and document their goals. Hence, the survey for this study was conducted through interview method. This method allows to collect the relevant data as required by the proposed assessment approach. Subjects were interviewed face-to-face within their natural working environment. With every single subject, an in-depth interview was conducted, which lasted three to six hours. Semi-structured interviewing was employed, in order to guarantee that the investigations were guided by the research questions, while keeping the flexibility to react on unforeseen subject responses and to explore unexpected phenomena.

Target audience of the interview survey. The target audience for this interview study were the stakeholders of software development projects that implemented and used traceability to support software lifecycle activities. The population of the target audience is large, because stakeholders of any software development are potential candidates.

Sampling strategy. Initially, a list of 85 potential companies was assembled. To prioritize this list, general information about each company was collected from the Internet and possible contact persons were identified. To select the suitable cases and subjects for this study, each potential case was prioritized. For this purpose, the framework proposed by Curtis et al. was applied [Curtis et al. 2000]. After prioritizing the list of

potential cases, the contact persons of highest prioritized cases were contacted in order to arrange an interview. Provided that the sampled subject agreed, either one or multiple interviews were conducted with key informants who are familiar with the company's software development process and its traceability practice.

Data collection. In advance, an interview guide was prepared, which comprises the following three parts: First, some general information about the subject and his or her working environment were collected. Each subject was asked to focus on a particular software development project for the remaining interview. Additionally, each subject was requested to select a project representative for the company's software development practice. Second, the subject was asked about the reasons for applying traceability in the concrete project. Thereby, detailed information on how and why every single project participant used traceability was collected. For this purpose, the aspects of a traceability usage scenario were recorded, such as actors, trace paths, artifacts, tools, tasks, and intention. Third, each subject was asked for important software process elements in the reported project, such as activities, tasks, actors, stakeholders, goals, artifacts, and tools to get a holistic view on the software process from beginning to the end. This part of the interview collected the data as described in Section 4.1, in order to identify traceability related requirements. Interview minutes and field notes were produced by a designated minute taker. Table 7.1 summarizes the characteristics of the participating subjects, their project and their company.

Data analysis. Qualitative content analysis [Schreier 2014] was applied to systematically extract relevant data from the interview minutes. The research question and the interview guide served as qualitative description model. A system of codes was derived for all three interview parts. These codes were used to classify all written interview minutes and field notes with the qualitative analysis tool MAXQDA10 [VERBI 2015]. After each interview two tasks were performed. First, a project specific TRM and TIM was extracted, based on the interview minutes. Therefore, the planning for purposed traceability method was applied (see Section 4.1 and 4.2). Second, the fitness for purpose of each project's traceability implementation was assessed. The assessment was carried out as described in Section 5.6. *Step 1:* the first step of the assessment procedure was skipped, because the subjects did not provide access to their development artifacts. *Step 2:* implemented artifact types, trace link types and trace path types were extracted from the interview minutes. *Step 3:* the implemented artifact types were linked to the required artifact types of the project specific TIM. *Step 4:* the implemented artifact types, trace link types, and trace path types were assessed for possible traceability problems.

Results. Table 7.2 summarizes the overall results of this study. As the projects were

Table 7.1.: Characteristics of the interviewed subjects, their project, and their company

Project Members	Company Employees	Case ID	Domain	Offering	Informant ID	Informant's Role	Informant's Experience [yr]
5..9	> 10,000	10	Insurance	Service	10.1	Process Manager	10..20
					10.2	Release Manager	5..10
10..100	1,001..10,000	9	IT Security	SW Product	9.1	Development Lead	10..20
					14	Avionic	HW Product
	< 100	16	Requirements Tool	SW Product	16.1	Development Lead	10..20
					3	Finance	Service
> 100	1,001..10,000	17	Telecommunication	HW Product	17.1	Process Manager	10..20
					6	Robotic	HW Product
< 5	100..1,000	8	Finance	SW Product	8.1	Project Manager	> 20
					2	Insurance	Service
	< 100	7	Finance	Service	7.1	Specification Manager	5..10
					13	Finance	Service
5..9	> 10,000	12	Retail	Service	12.1	Portfolio Manager	5..10
					12.2	Test Manager	10..20
					5	E-Commerce	SW Product
> 100	100..1,000	5	Logistic	Service	15.1	Business Analyst	> 20
					15	Logistic	Service
< 5	< 100	1	Public Service	Service	1.2	Project Manager	10..20
					4	Retail	SW Product
5..9	< 100	11	Insurance	Service	11.1	Development Lead	10..20

7.4. Study 3: Regulated Traceability Implementations

assessed at the type level only (artifact type, trace link type, and trace path type), the depiction of traceability problems related to the entities artifact, trace link and trace path were omitted in Table 7.2. None of the studied cases suffered from missing artifact type problems. Missing trace link types and missing trace path types were identified in all assessed projects. Two projects implemented superfluous trace link types as well as superfluous trace path types. Wrong trace link types and wrong trace path types were not found in any project.

Table 7.2.: Results of study 2: assessment results across the 17 studied cases

Traceability problem	Case ID																
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
\mathcal{M}_A - missing artifact type	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
\mathcal{M}_L - missing trace link type	3	3	4	6	6	4	3	3	2	1	2	5	8	2	7	3	5
\mathcal{M}_P - missing trace path type	5	6	6	9	9	7	4	5	2	1	3	6	12	3	9	4	7
\mathcal{S}_L - superfluous trace link type	0	2	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0
\mathcal{S}_P - superfluous trace path type	0	2	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0
\mathcal{W}_L - wrong trace link type	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
\mathcal{W}_P - wrong trace path type	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

7.4. Study 3: Regulated Traceability Implementations

This case study describes the application of the traceability planning and assessment approach in the domain of safety critical software development. This study was partly published in [Rempel et al. 2014].

Study objectives. In this study, the proposed traceability assessment approach was evaluated with respect to the Research Question 5, 6, and 7 (see Section 7.1).

Three study objectives can be derived from the two research questions. To answer Research Question 5, the assessment approach needs to be evaluated for its capability to determine whether or not the traceability implementation of a regulated software project complies with the relevant guidelines. To answer Research Question 6, the support for guideline migration needs to be evaluated. The capability of conducting traceability assessments in an automated manner needs to be evaluated for answering Research Question 7.

Study instrument. To explore the capabilities of the proposed assessment with respect to its automation, the approach needs to be directly applied to projects' traceability implementations. Case study research provides a suitable methodology with respect to these objectives [Runeson and Höst 2009]. The focus of this case study was traceability

7. Evaluation

implementations of safety-critical software projects that need to comply with one or multiple safety guidelines. Each project represents one unit of analysis.

Sampling strategy. In this study, a combination of the following two sampling strategies was chosen. First, cases from multiple domains (e.g. automotive and avionics) were chosen, which aimed for compliance with the different domain specific safety guidelines, in order to have a context where the results are generalizable. Second, only cases were included that aimed for compliance with multiple safety guidelines to ensure that the migration scenario is supported.

Data collection. Two sources information were identified for this study. First, the safety guidelines were used to derive required traceability information as a TIM (see Section 4.2). Second, the TID of the software projects were used for the assessment.

Throughout this case study, three safety guidelines, applicable for the development of safety-critical software in different domains, were analyzed. Table 7.3 shows these analyzed safety guidelines and each of their application domains. It outlines the number required traceability information. It provides an impression of the complexity of the described development process per guideline. All three guidelines were modeled following the traceability planning approach described in Chapter 4.

Table 7.3.: Characteristics of the formalized safety guidelines

	Safety guideline	Domain	Required traceability		
			$\mathcal{R}_{\mathcal{A}}$ ¹	\mathcal{R}_L ²	$\mathcal{R}_P \setminus \mathcal{R}_L$ ³
I	[DO-178B]	Aviation	31	3	5
II	[ISO 26262-6:2011]	Automotive	21	2	2
III	[ECSS 2009]	Space	73	6	2

¹required artifact types, ²required trace link types, ³required trace path types with more than one step

Traceability implementation data were assessed for four different software development projects. TOPCASED-SAM is a subproject of the TOPCASED initiative [Farail et al. 2006], which provides a set of modeling, transformation and verification tools for functional structured analysis [TOPCASE-SAM 2015]. *Relevant guidelines*: ISO 26262, ECSS-E-40, DO-178B; *Artifacts*: 123 requirements, 14 designs, 23 test cases, 15 test results, 1018 classes; *Relevant trace links*: 250. TOPCASED-REQ is a subproject of the TOPCASED initiative [Farail et al. 2006], which develops a generic, tool independent way for ensuring traceability between requirements and model elements and for managing requirements [TOPCASE-REQ 2015]. *Relevant guidelines*: ECSS-E-40, ISO 26262, DO-178B; *Artifacts*: 58 requirements, 9 designs, 6 test cases, 6 test results, 638 classes; *Relevant trace links*: 56. GeneAuto is an open-source toolset for converting Simulink,

Table 7.4.: Characteristics of the assessed software projects

Project	Relevant guideline(s)	Implemented traceability	
		I_A^1	I_L^2
TOPCASED-SAM	DO-178B, ISO 26262, ECSS-E-40	1,193	250
TOPCASED-REQ	DO-178B, ISO 26262, ECSS-E-40	717	56
GeneAuto	DO-178B	537	209
RAMI	DO-178B	443	553

¹implemented artifacts, ²implemented trace links

Stateflow, and Scicos models into executable program code [Gene-Auto 2013]. C code output is supported, Ada output is under development. *Relevant guideline*: DO-178B; *Artifacts*: 69 requirements, 216 design description artifacts, 240 source code artifacts, 12 test cases; *Relevant trace links*: 209. Rate Adjustment by Managing Inflows (RAMI) develops a TCP/IP flow control module for the Linux kernel and a suite of network evaluation utilities [RAMI 2015]. *Relevant guideline*: project dependent; *Artifacts*: 48 requirements, 120 design description artifacts, source code functions 275, test cases not accessible; *Relevant trace links*: 553.

Data analysis. The collected traceability implementation data of the studied cases were assessed for their compliance with the traceability that is prescribed by relevant safety guidelines. Therefore, the proposed assessment procedure, along with the PurISTA prototype, was used to quantify the number of traceability problems per studied case with respect to the relevant guideline as shown in Table 7.4.

Results. The projects were assessed for safety guideline compliance. Traceability problems that indicate superfluous traceability entities are not relevant for compliance assessments, because a superfluous entity does not violate the required traceability information. It would only be an indicator that the implementation is not cost-effective. For this reason, problems related to superfluous elements were omitted from the assessment. Table 7.5 shows the aggregated results of the projects versus guidelines traceability assessment. The first and second column refer to the project and the guidelines that were compared. As this study focus on investigating the compliance between project specific traceability implementation and traceability prescribed by safety guidelines, only those traceability problems related to the quality attributes completeness and correctness (see Section 5.4) were investigated. Columns three to six refer to the identified traceability problems that were found during the assessment. Column three summarizes the number of missing artifact types ($\mathcal{M}_{\mathcal{A}}$). Column four shows the aggregated number of missing

7. Evaluation

trace link types (\mathcal{M}_L) and missing trace path types (\mathcal{M}_P), while the entailed aggregated number of missing trace links (\mathcal{M}_L) and missing trace paths (\mathcal{M}_P) is shown in brackets. The aggregated number of missing trace links (\mathcal{M}_L) and missing trace paths (\mathcal{M}_P) for implemented trace link types and implemented trace path types is shown in column five. Column six shows the aggregated number of wrong trace link types (\mathcal{W}_L) and wrong trace path types (\mathcal{W}_P), while the aggregated number of entailed wrong trace links (\mathcal{W}_L) and wrong trace paths (\mathcal{W}_P) is shown in brackets.

Table 7.5.: Results of study 3: overview of assessment results per project

Assessment		Identified traceability problems			
Project \longleftrightarrow Guideline		\mathcal{M}_A^1	$\mathcal{M}_L \cup \mathcal{M}_P^2$ $\downarrow (\mathcal{M}_L \cup \mathcal{M}_P^3)$	$\mathcal{M}_L \cup \mathcal{M}_P^4$	$\mathcal{W}_L \cup \mathcal{W}_P^5$ $\downarrow (\mathcal{W}_L \cup \mathcal{W}_P^6)$
TC-SAM	DO-178B	4	7 (3,480)	8	0 (0)
	ISO 26262-6	3	2 (1,268)	45	0 (0)
	ECSS-E-40	1	5 (2,316)	8	0 (0)
TC-REQ	DO-178B	4	7 (2,115)	0	0 (0)
	ISO 26262-6	3	2 (757)	0	0 (0)
	ECSS-E-40	1	5 (1,413)	0	0 (0)
GeneAuto	DO-178B	3	7 (1,383)	167	0 (0)
RAMI	DO-178B	3	7 (1,377)	74	0 (0)

¹missing artifact types, ²missing trace link types or trace path types, ³entailed missing trace links or trace paths, ⁴missing trace links or trace paths for implemented trace link types or trace path types, ⁵wrong trace link types or trace path types, ⁶entailed wrong trace links or trace paths

The TIMs that were derived from multiple guidelines were compared to identify differences with respect to the required traceability information. The assessment results are shown in Table 7.6. The first two columns show the compared guidelines. Column three quantifies the number of required artifact types, which were required by the first but not by the second guideline. Column four quantifies the number of required artifact types that the two guidelines had in common. Column five quantifies the number of required artifact types, which were required by the second but not by the first guideline. The columns six to eight repeat these comparisons with respect trace path types. This comparison includes trace links, because the set of all required trace path types contains all required trace link types. The results demonstrate that differences among the guidelines are relatively small in terms of required traceability. However, these differences are required to be addressed if a project needs to be migrated to a different or revised guideline.

Table 7.6.: Results of study 3: comparison of guidelines with respect to required artifact types and required trace paths

Guideline Comparison $I \longleftrightarrow II$	$\mathcal{R}_{\mathcal{A}}^1$			$\mathcal{R}_{\mathcal{P}}^2$		
	$I \setminus II$	$I \cap II$	$II \setminus I$	$I \setminus II$	$I \cap II$	$II \setminus I$
ISO 26262 DO-178B	0	6	2	0	4	4
DO-178B ECSS-E-40	0	8	1	0	8	1

¹required artifact types, ²required trace path types

7.5. Study 4: Traceability Assessment Results

This section describes a survey that was carried out with participants of the assessed safety projects and safety certifiers to evaluate the usefulness of the generated assessment results.

Study objectives. In this study, the assessment results of study 3 were evaluated with respect to the Research Question 5 and 3 (see Section 7.1). Two study objectives can be derived. To answer Research Question 5, the generated assessment results need to be evaluated for its usefulness to assess the compliance of a project's traceability implementation to a guideline. The main concern of safety guidelines is to ensure that the safety analysis activity can be executed. This means that, in order to answer Research Question 5 with respect to safety-critical software development, it is necessary to evaluate if the assessment results can be used to determine if a safety analysis is feasible with a project's traceability implementation.

Study instrument. Project participants of the assessed projects and experienced safety certifiers can provide reliable judgments with respect to the study objectives. The above stated study objectives demand for a descriptive survey. The survey for this study was conducted through questionnaire method with standardized Liker scales [Kitchenham and S. L. Pfleeger 2008]. This method allows producing quantitative survey results. Based on these results, averages and trends can be determined with respect to the opinions of the project participants and certifiers.

Target audience of the questionnaire survey. The study objectives reduced the number of potential survey participants notably. The target audience are project participants of study 3 (see Section 7.4) and safety certifiers. Due to this constraint, the population of the target audience is relatively small.

Sampling strategy. The following three strategies were applied to find safety project participants and certifiers that answered the questionnaire. First, members of the assessed projects were contacted. Second, safety certification professionals known through social networks were asked for participation. Third, the snowball sampling technique

7. Evaluation

[Biernacki and Waldorf 1981] was applied. Study participants who returned the questionnaire were asked if they knew other safety project member or safety certifier.

Data collection. The qualitative feedback on the assessment results was collected via a questionnaire, which comprised the following four parts:

1. The first part (Preliminaries) provided preliminary information how the questionnaire is structured, contact information, anonymity assurance, and which questions are expected to be answered.
2. To remove ambiguity, elementary concepts such as traceability, functional safety, and software safety guidelines were defined and explained in the Introduction part.
3. The third part (Preliminary questions) asked preliminary questions about the professional background of the study participant. The questions about the familiarity with functional safety and the participation in projects that were aiming for functional safety served as the inclusion criteria for this study. Only participants who confirmed familiarity with functional safety and the participation in a safety-critical project were included to the results.
4. The main part of the questionnaire (Questions about the assessment) asked the study participants for their opinion on the usefulness of the assessment results of five traceability problems (missing trace link, missing trace link type, missing artifact type, incorrect trace path, and incorrect trace link). For each problem, an illustrating example was provided in textual and graphical form. Additionally, an assessment report excerpt from case C was provided for each problem. The participants were asked to indicate their opinion on a Likert scale *strongly agree*, *agree*, *neither agree nor disagree*, *disagree*, and *strongly disagree* [Kitchenham and S. L. Pfleeger 2008]. The first statement claimed that the assessment results indicate *non-compliance* of traceability with a guideline. The second statement claimed that the assessment results indicate a safety risk. The third statement asked, whether or not, the assessment results are helpful for a certifier. The fourth statement asked, whether or not, the assessment results are helpful for a project member in a safety project.

The complete questionnaire is provided in Appendix B. Before distributing the questionnaire to the project participants and certifiers, its content was verified and improved, following an iterative approach. First, other PhD students were asked to provide critical feedback on the content of the questionnaire in terms of suitability and understandability. Second, two persons who worked in safety-critical development projects were asked to fill-out the questionnaire. After the questionnaire was completed, each person was interviewed for one hour to further improve the suitability and understandability of

7.5. Study 4: Traceability Assessment Results

the questionnaire. Main objective of this second iteration was to pretest the developed questionnaire. Accordingly, all results of this iteration were excluded from the study results. This iteration was also used to measure, whether or not the estimated duration of 40 minutes was sufficient to fill-out the questionnaire. Third, 51 safety project participants and certifiers were contacted over a period of three months. After this period, 17 subjects returned a completely filled questionnaire that fulfilled the study inclusion criteria. The participating subjects had an average overall professional experience of 15.29 years (std: 9.58, min: 4, med: 13, max: 45). The vast majority had great expertise in the field of safety-critical software systems (expert: 7, more than two years experience: 8, less than two years experience: 2). All subjects either participated in a safety project or certified a safety project (participated in safety project: 16, participated in certified project: 11, certified a project: 6). Each subject was asked to indicate his or her primary project role as well as one or many secondary roles in the project. As depicted in Figure 7.2, the subjects held a great variety of roles such as *developer*, *manager*, *architect*, *tester*, *analyst*, *auditor*, *consultant*, *administrator*, *trainer*, *safety engineer*, and *certification engineer*. As shown in Figure 7.3, the subjects came from a wide range of different domains, namely *aviation*, *space*, *military*, *automotive*, *medical*, *energy*, *agriculture*, *railway*, and *industrial automation*.

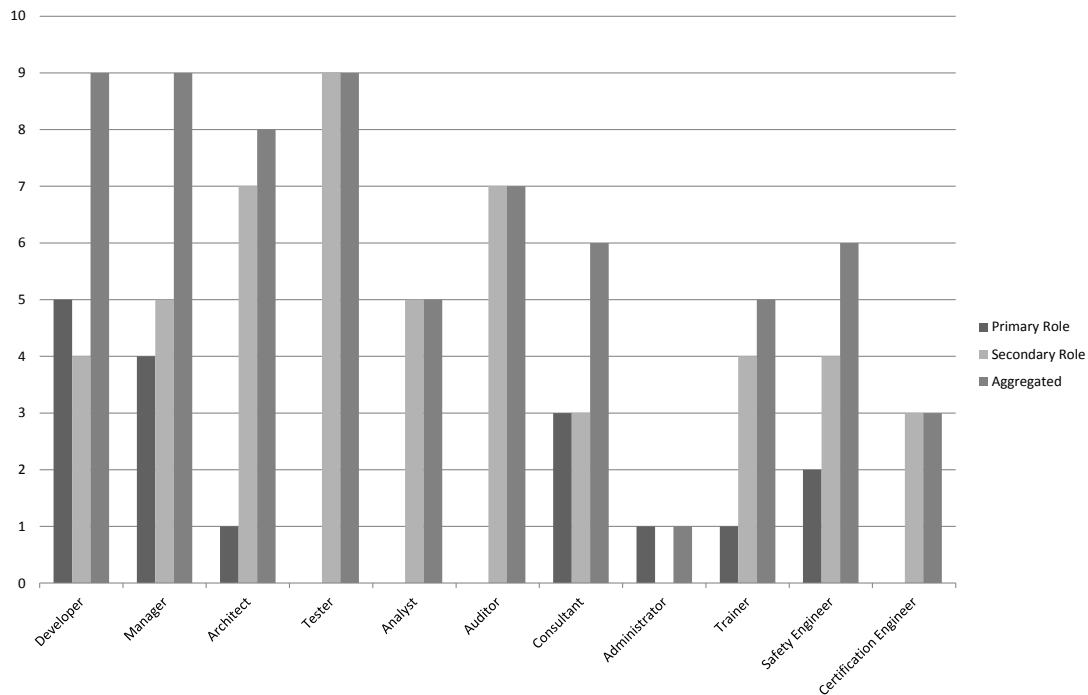


Figure 7.2.: Roles of the subjects in study 4

Data analysis. The returned questionnaires data were analyzed to quantify how useful

7. Evaluation

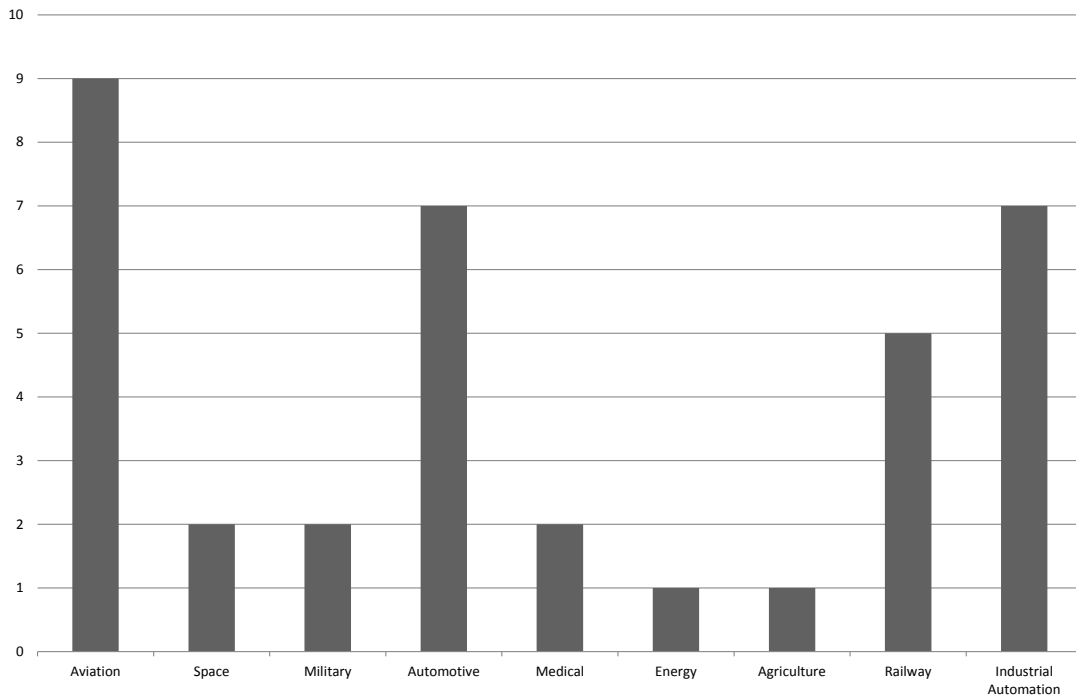


Figure 7.3.: Safety domains of the subjects in study 4

safety project participants and certifiers perceive the assessment results. Therefore, each of the subjects' answers was converted into a number. Subjects were asked to indicate their opinion on various statements and questions using an agreement scale *strongly agree*, *agree*, *neither agree nor disagree*, *disagree*, and *strongly disagree* [Kitchenham and S. L. Pfleeger 2008]. Each opinion was coded by the following scheme: *strongly agree* = 2, *agree* = 1, *neither agree nor disagree* = 0, *disagree* = -1, *strongly disagree* = -2. Based on these opinion scores, aggregated values over all subjects were calculated.

Results. Figure 7.4 compares the subjects' agreement or disagreement to the claim that the reported assessment result is an indicator for traceability non-compliance with the safety guideline. As stated above, opinions are coded as *strongly agree* = 2, *agree* = 1, *neither agree nor disagree* = 0, *disagree* = -1, *strongly disagree* = -2. For each of the four reported traceability problems (missing artifact type, missing trace link type or trace path type, incomplete trace link type or trace path type, and wrong trace link type or trace path type), the 17 subjects agreed or strongly agreed that the assessment result is an indicator for traceability non-compliance with the safety guideline. Individual statistics are as follows: missing artifact types (avg: 0.94, med: 1, std: 1.19), missing trace link type or trace path type (avg: 1.59, med: 2, std: 0.79), incomplete trace link type or trace path type (avg: 1.12, med: 2, std: 1.27), wrong trace link type or trace path type (avg: 0.59, med: 1, std: 1.37).

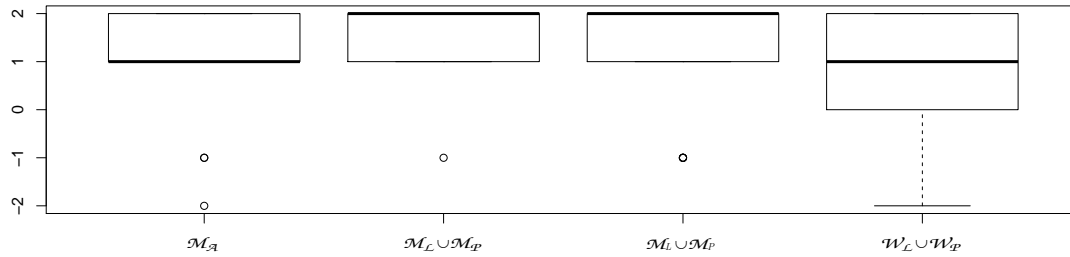


Figure 7.4.: Results of study 4: subjects' opinions on the traceability assessment results with respect to indicate compliance problems

Figure 7.5 compares the subjects' agreement or disagreement to the claim that the reported assessment result is an indicator for a safety risk within the project. For each of the four reported traceability problems, the 17 subjects agreed or strongly agreed that the assessment result is an indicator for a safety risk in the project. Individual statistics are as follows: missing artifact types (avg: 0.65, med: 1, std: 1.06), missing trace link type or trace path type (avg: 1.41, med: 2, std: 0.79), incomplete trace link type or trace path type (avg: 1.18, med: 2, std: 1.13), wrong trace link type or trace path type (avg: 0.12, med: 1, std: 1.32).

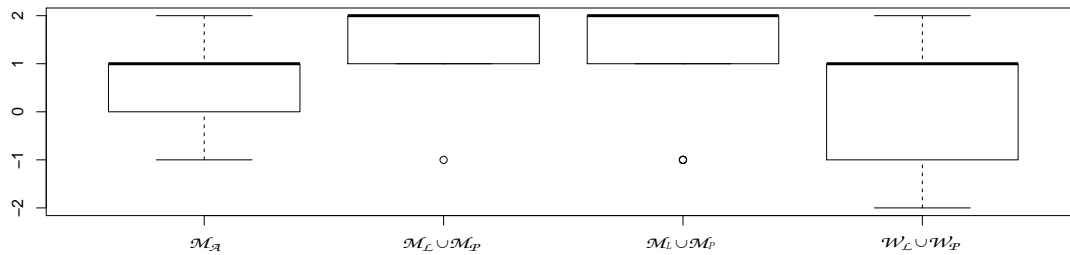


Figure 7.5.: Results of study 4: subjects' opinions on the traceability assessment results with respect to indicate safety problems

Figure 7.6 compares the subjects' agreement or disagreement to the claim that the reported assessment result is helpful for a certifier to check the compliance of a project's traceability with a safety guideline. For three of the four reported traceability problems, the 17 subjects agreed that the assessment result is an indicator for a safety risk in the project. For one of the reported traceability problems, the 17 subjects neither agreed nor disagreed. Individual statistics are as follows: missing artifact types (avg: 0.29, med: 1, std: 1.04), missing trace link type or trace path type (avg: 0.76, med: 1, std: 1.15), incomplete trace link type or trace path type (avg: 0.82, med: 1, std: 1.19), wrong trace link type or trace path type (avg: 0.24, med: 0, std: 1.2).

Figure 7.7 compares the subjects' agreement or disagreement to the claim that the reported assessment result is helpful for a project participant to ensure the compliance of a project's traceability with a safety guideline. For three of the four reported traceability

7. Evaluation

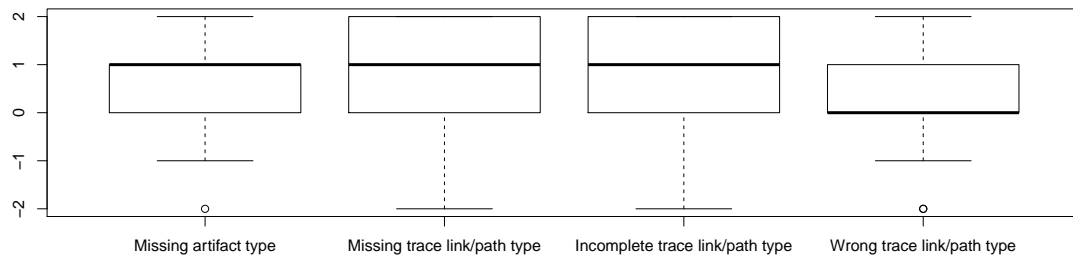


Figure 7.6.: Results of study 4: certifiers' opinions on the usefulness of the traceability assessment results for their work

problems, the 17 subjects agreed that the assessment result is an indicator for a safety risk in the project. For one of the reported traceability problems, the 17 subjects neither agreed nor disagreed. Individual statistics are as follows: missing artifact types (avg: 0.41, med: 1, std: 1.12), missing trace link type or trace path type (avg: 1, med: 1, std: 1.72), incomplete trace link type or trace path type (avg: 0.94, med: 1, std: 1.03), wrong trace link type or trace path type (avg: 0.24, med: 0, std: 1.3).

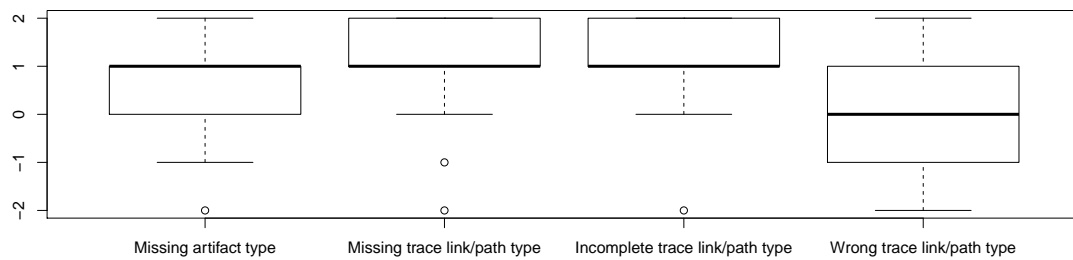


Figure 7.7.: Results of study 4: project participants' opinions on the usefulness of the traceability assessment results for their work

7.6. Discussion

This section discusses the findings the conducted studies (see Sections 7.2 and 7.4) with respect to the stated research questions.

7.6.1. Research Question 1: Relevance

The results concerning the relevance of each traceability problem (see Figure 7.1) show a clear trend. Problems related to the quality attributes *completeness* and *correctness* are consistently rated as more relevant than problems related to the quality attributes *appropriateness*. A possible reason for this disparity could be the difference in the problem implication, depending on the quality attribute. On the one hand, problems related to the quality attribute *appropriateness* imply that unnecessary effort was spent to im-

plement superfluous traceability data. Addressing these problems avoids unnecessary effort and save costs. On the other hand, problems related to the quality attributes *correctness* imply that decisions are made on wrong assumptions and problems related to *completeness* imply that either important artifacts or activities were missed. As found within the safety case study in Section 7.4, within the context of safety critical software development, these problems represent a potential safety risk. Comparing the potential threat of safety risks with the potential threat of monetary risks, the observed relevance disparity between the quality attributes appears logical.

The results also show traceability implementation data specific trends. For the quality attributes *completeness* and *correctness*, problems related to traceability implementation data types (artifact type, trace link type, and trace path type) are rated as more relevant than problems related to traceability implementation data instances (artifact, trace link, and trace path). A possible reason for this disparity could be the different root cause of these problems. While a problem related to traceability types typically indicates a systematic failure within the development process, a problem related to traceability instances typically indicates an oversight. For example, completely missing traceability between high-level and low-level requirements is an indicator for the fact that the activity to implement traceability between these two artifact types was completely missed out, which would be reported as a missing trace link type. By contrast, for example, a single low-level requirement, which is not traceable to a high-level requirement, indicates a shortcoming of an executed traceability implementation activity. Although, both problems indicate a potential safety risk and are rated relatively high compared to other quality attributes, the impact of a systematic failure within the development process is potentially even higher than an individual oversight. Also, after detecting a problem, the effort to fix such a systematic failure is much higher than fixing a single oversight.

For the quality attribute *appropriateness*, problems related to traceability implementation data types (artifact type, trace link type, and trace path type) are rated as less relevant than problems related to traceability implementation data instances (artifact, trace link, and trace path). A possible reason for this disparity could be the different level of detail. As described above, problems related to this quality attribute indicates unnecessary effort. While an appropriateness problem related to traceability types only indicate that unnecessary effort was spent, appropriateness problems at the instance level can be used to explicitly quantify the number of superfluous trace links that were implemented.

7.6.2. Research Question 2: Completeness

According to the feedback collected with a traceability experts survey (see Section 7.2), 11 subjects agreed with the assumption that the proposed assessment model covers all possible traceability problems with respect to the fitness for purpose of a project's

7. Evaluation

traceability implementation. One subject argued that the support of bidirectional trace link types is missing. Since trace link types and trace path types are directed in the TAM and TIM, each trace link type or trace path type has a source and a target (see Section 3.2 and 4.2.1). A bidirectional required trace link type between two artifact types \mathcal{A}_1 and \mathcal{A}_2 would be specified by the following two trace link types: $\mathcal{A}_1 \rightarrow \mathcal{A}_2$ and $\mathcal{A}_2 \rightarrow \mathcal{A}_1$. If a trace link type was implemented unidirectional only, e.g. $\mathcal{A}_1 \rightarrow \mathcal{A}_2$, a missing trace link type would be detected by the assessment criterion of $\mathcal{M}_{\mathcal{L}}$. Hence, bidirectional link types are covered by the proposed traceability problem classification.

This result leads to the finding that the proposed assessment approach is based on a complete problem classification with respect to its objective. The potential risks of producing incomplete assessment results are mitigated. This complete coverage of all possible structural traceability problems is an important contribution to the traceability community. For the first time, a comprehensive traceability assessment method is presented. Recently, Merilinna and Pärssinen lamented the fact that no comprehensive traceability assessment method exists [Merilinna and Pärssinen 2010], which emphasizes the relevance of this contribution.

7.6.3. Research Question 3: Feasibility of Software Lifecycle Activities

Traceability is required by numerous software lifecycle activities such as, change impact analysis, coverage analysis, and compliance verification. As demonstrated in study 2, the feasibility of these software lifecycle activities can be determined with the proposed assessment approach. The planning for a purposed traceability method (see Chapter 4) can be used to identify traceability goals and derive required traceability information as TIM. This required traceability information specifies the target state of a traceability implementation. If a project fails to implement complete and correct traceability with respect to this target state, the activities that require these missing or wrong traceability implementation data cannot be supported. Traceability problems with respect to completeness were identified in all 17 assessed cases in study 2. The assessment approach identified activities in all projects, which were essential to the project but not feasible due to missing trace link types or missing trace path types.

Determining the feasibility of a safety analysis is essential for safety-critical software systems. To argue the safety of a developed system, a safety case is compiled as a result of a safety assurance process [Zeller et al. 2014]. The safety argument needs to make clear that all safety risks were identified and to demonstrate how these risks were mitigated. To demonstrate that all safety requirements have been validated, satisfied, and realized, and that their origin is documented, complete and correct, traceability is required for all safety requirements. In study 4 (see Section 7.5), safety-critical project participants and certifiers agreed that the generated results of the proposed assessment approach are suitable to determine the feasibility of the safety analysis activity. As

summarized in Figure 7.5, the safety project participants and certifiers agreed that the traceability problems are an important indicator for a safety risk within the project. They agreed that the identified traceability problems indicate that the required traceability implementation data for the safety argument were missing. A case study of an emergency brake system at Daimler Chrysler came to similar findings [Ridderhof et al. 2007]. Thereby, a safety case was created by assessing traceability for missing links with constraints formulated in OCL. However, it suffers from two shortcomings. First, the assessment focuses on missing trace links only, which does not cover all possible traceability problems as discussed in Section 7.6.2. Second, instead of providing a general traceability assessment approach, the assessment procedure was specifically developed for the automotive domain, which impedes its generalizability.

7.6.4. Research Question 4: Cost-effective Implementation

The manual creation and maintenance of trace links is cost-intensive [Cleland-Huang et al. 2004; Heindl and Biff 2005]. Implementing superfluous TID should be avoided to ensure cost effective traceability implementations. In study 2, the application of the traceability assessment approach was demonstrated to detect superfluous TID. These detected superfluous TID are relevant for the cost-effectiveness, as unnecessary maintenance effort is avoided. This study also demonstrated its support for creating cost-effective traceability implementation. The applied planning method supports the specification of a TIM that does not require the implementation of any superfluous traceability data, which would imply unnecessary traceability creation effort.

7.6.5. Research Question 5: Compliance

As described in Section 7.4, a comprehensive case study was conducted with software projects aiming to comply with safety guidelines. Thereby, the proposed traceability planning and assessment approach was applied, which demonstrated its practical applicability. This applicability of the planning and assessment approach was an important finding of the case study with respect to the industrial usage potential of the approach.

The traceability assessment results in Table 7.5 show that projects mainly struggled with the traceability completeness. This results confirm earlier non-systematic traceability compliance assessments [Mäder et al. 2013; Panesar-Walawege et al. 2010].

The results of the survey with safety project participants and certifiers in study 4 (see Section 7.5) suggests that the assessment approach is useful for both parties. As summarized in Figure 7.7, safety project participants confirmed that the assessment and reporting of traceability problems with respect to the quality attribute *completeness* is useful for their daily work. This result is anything but surprising. As elaborated in Section 7.6.1, a safety argument can only be based on complete traceability. Similarly to the project participants, certifiers agreed that the assessment and reporting of traceabil-

7. Evaluation

ity problems with respect to the quality attribute *completeness* is useful for their daily work, as summarized in Figure 7.6. The similarity in how the usefulness of the traceability assessment results is perceived by project participants and certifiers emphasizes the similarity of concerns from different perspectives. Both parties are concerned about the safety of the developed system. The project participant perspective is to build a coherent safety argument. The certifiers perspective is to judge the coherence of the provided arguments. As suggested by the results of study 3, reporting exiting traceability problems with respect to the completeness is a useful tool for both perspectives.

7.6.6. Research Question 6: Migration

When an existing product is introduced into a new market, it may be necessary to certify the product under a new guideline. Migrating to a new guideline implies the risk that new traceability requirements needs to be addressed. The planning approach can be used to support these migration scenarios. Multiple TIMs can be derived for multiple guidelines. As demonstrated in study 3, these multiple TIMs can be compared for deviating traceability requirements. Based on these results, the migration of the existing traceability implementation can be planned accordingly.

7.6.7. Research Question 7: Continuous Assessment

The case study conducted in Section 7.4 demonstrated a main contribution of the assessment approach. Due to the PurISTA tool (see Chapter 6), the traceability assessment can be executed automatically with TID of an entire project. The solution is not limited to specific artifact types. Especially the cases where artifact types comprised hundreds or thousands of artifacts that had to be assessed, in order to check the compliance of the implemented traceability, illustrated the high practical potential of the automated compliance assessment. The PurISTA tool can analyze the traceability network faster than humans can do. This automatic assessment approach can be scaled up to millions of artifacts as demonstrated in [Rempel and Mäder 2015b]. Manual traceability assessments, as conducted in [Mäder et al. 2013], would be impossible for this project sizes. In [Panesar-Walawege et al. 2010], a case study specific traceability assessment approach was implemented, which supports automatic traceability assessment. However, the approach was limited to two specific trace link types and a specific guideline (IEC 61508 [IEC 61508:2010]). A comprehensive conceptual model for the traceability assessment was missing.

7.6.8. Limitations

Despite the argued benefits of the proposed traceability planning and assessment approach, it also has some limitations, which will be discussed in this section.

As discussed in Sections 7.6.3 and 7.6.5, the assessing and reporting traceability problems are useful to build a coherent safety argument. An automatically executed assessment approach, as demonstrated in study 3 (see Section 7.4) with the application of the PurISTA prototype (see Section 6.1.5), can be used to replace time consuming manual assessments and support incremental approaches due to its fast repeatability. However, the final safety argument is compiled by humans [Storey 2010]. Although the automated assessment approach supports the compilation of a safety argument, a safety argument still requires final expert’s engineering judgment.

Another important limitation of the implemented PurISTA prototype (see Section 6.1.5) was highlighted by several subjects of the survey in study 3. They argued that users of the traceability assessment tool need to trust the correctness of the assessment result. The ultimate approach for establishing trust would be to qualify the traceability assessment tool, which has not yet been achieved. At the present moment, trust can only be established through the demonstration in case studies.

As discussed in Section 6.1.3, the collector component of the PurISTA prototype supports a variety of existing artifact management tools and formats. However, a full coverage of all the existing formats is not realistic. Therefore, depending on the project specific representation of the traceability implementation data, additional development effort may be required to automatically parse existing artifacts and trace links. A potential solution to this limitation could be the OSLC guideline [OASIS 2015], which aims to establish a unified communication protocol for tools that manage software life-cycle artifacts. Once such a guideline is supported by all software engineering tools, this current limitation would disappear.

7.7. Threats to Validity

In this section, the threats to the validity of the presented study will be discussed as well as how these threats have been mitigated. The discussion is organized by the types of validity. Therefore, a common validity classification scheme [Runeson and Höst 2009; Yin 2009] is applied, which distinguishes the following four types of threats to validity:

- *Construct validity*: The degree to which the operational measures that are studied really represent what the researchers have in mind and what is investigated according to the research questions (see Section 7.7.1).
- *External validity*: The degree to which the findings of the study can be generalized to the studied population and to other research settings (see 7.7.2).
- *Internal validity*: The degree to which a causal effect of the independent on the dependent variable can be concluded (see 7.7.3).

7. Evaluation

- *Reliability*: The degree to which the empirical results depend on the researcher conducting the study (see 7.7.4).

7.7.1. Construct Validity

As an important part of the proposed traceability planning and assessment approach, a classification of traceability problems was introduced (see Section 5.4). As demonstrated within the safety cases study (see Section 7.4), this classification can be used to measure the fitness for purpose of a traceability implementation with respect to the quality attributes *completeness*, *correctness*, and *appropriateness*. To mitigate the threat that those operational measures are not suitable for the purpose of assessing the suitability of traceability, the set of measures was qualitatively evaluated in two steps. First, for each proposed measure, feedback was collected from 13 traceability experts. The results in Figure 7.1 suggest the agreement of the experts with the validity of the proposed measures. Second, 17 safety domain experts were asked to provide their opinion on the suitability of the proposed measures with respect to the domain specific goals *safety* and safety guideline *compliance*. The results in Figure 7.5 and Figure 7.4 suggest the agreement of the safety domain experts with the validity of the proposed measures to indicate safety and compliance risks within safety critical software projects. Additionally, one of the proposed measures, namely, missing trace links was used in similar safety related case studies [Ridderhof et al. 2007; Panesar-Walawege et al. 2010; Mäder et al. 2013] for assessing the implemented traceability.

7.7.2. External Validity

For this work, external validity concerns whether the observed findings and drawn conclusions from the case study consisting of four cases and three safety guidelines hold over variations in guidelines and projects. The fact that the studied cases diverge across multiple guidelines and industrial projects of various sizes and domains suggests that the approach is applicable across a wide variety of projects. However, a larger multi-domain and longitudinal evaluation is required to draw final conclusions.

7.7.3. Internal Validity

In both studies (see Section 7.2 and 7.4), quantitative feedback was collected from either traceability experts or safety domain experts through a questionnaire survey. The potential threat of any subject's misunderstanding about the underlying concepts of a question was addressed by the following countermeasures. First, a comprehensive briefing section was added to both questionnaires, which in each case was extensively tested with pilot participants, who were excluded from the results. The pilot participants were independently interviewed to eliminate potential shortcomings in the ques-

tionnaire. Second, within the questionnaire that was answered by the safety domain experts illustrating examples of the traceability problem classifications were provided from the project GeneAuto [Gene-Auto 2013] and the safety guideline DO-178B [DO-178B]. Third, additional qualitative feedback was collected from all subjects to identify potential misunderstandings. For this purpose, a free text form was provided next to each question, where the subject could explain his or her reasoning, which was extensively used by the subjects. This qualitative feedback also provided valuable insights on the limitations of the proposed approach (see Section 7.6.8).

For the case study in the safety context, another potential threat exists due to the required studying and understanding of the safety guidelines and the creation of the respective prescribed traceability model. The responsible person may misunderstand or entirely miss requirements of a guideline. For mitigation purposes, three countermeasures were taken. First, the same traceability planning procedure (see Section 4) was followed all analyzed guidelines, searching for the concepts: traceability goals, traceability usage tasks, traceability implementation tasks, required artifact types, required trace link types, and required trace path types. Second, the four eyes principle was applied. Thereby, all safety guideline models used for the safety case study were reviewed by a colleague with extensive experience in the development of safety-critical software.

To automatically assess the cases' traceability implementation data, the traceability collector of the PurISTA prototype had to be adapted according to the project specific structure. The artifacts were typically diverse and often spread across multiple tools and repositories. To mitigate the risk that all traceability implementation data were collected correctly, the structure of each project and its available artifacts was carefully examined. Additionally, completeness and consistency checks of the generated results were performed, where possible. However, due to the amount of assessed data, which prevents complete coverage of the manual verification procedure, there remains a potential risk that artifacts or trace links may have missed or misclassified.

7.7.4. Reliability

The reliability threat concerns whether a replication of the safety related case study would produce similar assessment results. A potential threat exists in the collection and preparation of the project data used to produce the assessment results. To avoid especially manual bias during the project data preparation and to ensure reproducible results, the process of data collection and assessment was fully automated. Due to the public availability of the project artifacts and the fully automated collection and analysis process, the study can be replicated and additional projects could be included to further broaden the data corpus.

8. Conclusions and Outlook

This final chapter summarizes the conclusions in Section 8.1 that can be drawn from this thesis. In Section 8.2, future work is identified and discussed, which is related to the contributions of this thesis. Potential directions to carry on with the presented work are outlined.

8.1. Summary

This thesis has presented an approach to assess the fitness for purpose of a project's traceability implementation. It supports the implementation of purposed traceability that can be trusted. The proposed approach addresses multiple shortcomings of existing work. Current traceability planning approaches do not sufficiently support the planning for purposed traceability. Existing definitional approaches lack a definition of the term fitness for purpose of a traceability implementation. Existing analytical approaches lack clear and comprehensive assessment criteria to support the performance of systematic and reproduceable assessments. Only two existing analytical approaches support the automated assessment of traceability. However, both approaches are limited to UML or SysML design artifacts. Other analytical approaches support manual assessments only.

The presented assessment approach consists of two important parts. First, a method to plan for purposed traceability was presented. This is a prerequisite for assessing the fitness for purpose of a project's traceability implementation. Second, a traceability assessment approach was proposed that compares the traceability implementation data of a project with the traceability implementation target state that was specified with the proposed planning method. The proposed TAM provides clear assessment criteria for detecting traceability problems with respect to the fitness for purpose. A questionnaire survey with traceability experts was conducted to evaluate the TAM's traceability problem classification for its completeness and each problem type for its relevance. The study results suggests that the TAM provides a complete classification of relevant problem types.

The proposed assessment approach provides support for non-safety-critical. It can be used for non-safety-critical projects to determine the feasibility of important software lifecycle activities. It can also be used to determine the cost-effectiveness of a project's traceability implementation. Furthermore, the planning cost-effective traceability implementations is supported.

8. Conclusions and Outlook

Safety-critical software projects are supported with their safety argument. The assessment approach provides detailed information about the traceability of safety requirements with respect to its completeness and correctness. The approach also supports the compliance assessment of a traceability implementation with respect to the safety guidelines that need to be followed. Migration scenarios to new or revised guidelines are supported by the purposed traceability planning approach. Different traceability requirements can be detected easily and provide important informations to prepare the migration.

The PurISTA tool provides support for automated traceability implementation assessments. Automated assessments are relevant for safety-critical and non-safety-critical projects. Especially in project environments with hundreds or thousands of artifacts, the execution of automated assessments is required to verify the fitness for purpose in a continuous manner.

8.2. Future Work

There is always potential to proceed and improve. This section outlines further research directions.

The main focus of the proposed approach was to detect traceability problems indicating traceability that is not suitable to achieve all traceability related goals of a project. Although, detecting those traceability problems is an important first step, further measures need to be taken to control the fitness for purpose of a project's traceability implementation. Other engineering disciplines introduced the idea of feedback loops in quality control systems [Deming 2000]. Central idea of this quality feedback loop is to continuously monitor the product quality. Detected problems are addressed by correcting the undesired deviations. A commonly applied feedback control approach is the Plan-Do-Check-Act (PDCA) cycle, which is recommended by the ISO 9000 standard [ISO 9001:2008]. Leveraging the assessment approach presented in this work, a traceability control system could be developed, which aims to automatically correct traceability to preserve its fitness for purpose.

The idea of establishing a traceability control system can be further extended. Existing traceability problems may be the effect of other root causes, such as process shortcomings. Detecting and eliminating these root causes can prevent the re-occurrence of detected traceability problems. Leading to sustained improvement of the traceability implementation process. Systematizing the root cause analysis of traceability problems and providing effective change methods can provide important benefits for projects, especially within the context of safety critical software.

Qualitative studies on the reason of structural traceability problems, such as missing trace links found that the implementation of traceability is often considered as too expensive [Arkley and Riddle 2005; Mäder et al. 2009b] with respect to its benefits.

At present, resilient and accepted traceability implantation cost models are lacking [Cleland-Huang et al. 2014]. Developing such cost models would provide project managers some important input for estimating traceability related cost. Combining such a cost model with the proposed assessment approach would further allow to quantify the impact of the identified traceability problems with respect to its expected rectification costs.

The proposed approach focused on the assessments of a project's fitness for purpose. The implemented traceability is assessed with respect to the traceability implementation target state. The semantic analysis of existing trace links could be a completely different direction for analytical assessment approaches. The detection of semantic defects could be used to identify contradictions, inconsistencies or other shortcomings related to trace link semantics. Similar to the problem of the automatic generation of trace links, machine learning approaches or information retrieval techniques could be useful to support the detection of semantic traceability defects.

List of Figures

2.1. Directed trace link between two artifacts, a source and a target artifact, that supports bi-directional traversal (adapted from [Gotel et al. 2012b])	8
2.2. Overview of the traceability lifecycle activities and their sub-activities	11
3.1. Meta-model of relevant Traceability Implementation Data (TID) characterizing the fundamental concepts of a project's traceability implementation	24
3.2. Overview of the traceability assessment approach	26
4.1. Meta-model of the Traceability Requirements Model (TRM)	33
4.2. Illustrating autopilot project: goals of the stakeholder Federal Aviation Administration (FAA) that were derived from the guideline DO-178B	35
4.3. The autopilot project: activities derived from the guideline DO-178B that are supposed to address the goals of the stakeholder Federal Aviation Administration (FAA)	36
4.4. The autopilot project: identified activities that require traceability (traceability usage activities) and their related goals (traceability goals)	37
4.5. The autopilot project: derived traceability requirements (traceability implementation activities)	40
4.6. Meta-model of the extended Traceability Information Model (TIM)	41
4.7. The autopilot project: deriving required trace path types from a traceability implementation activity	43
4.8. The autopilot project: deriving required trace link types from required trace path types and software lifecycle activities	44
4.9. The autopilot project: chain of evidence to justify the purpose of the required trace link type HLR -> LLR	46
5.1. Overview of the traceability lifecycle including the new traceability verification activity	49
5.2. Meta-model of the Traceability Assessment Model (TAM)	50
5.3. Overview of the traceability implementation elements that are subsumed by the TAM element traceability entity	51
5.4. Quality tree of the high-level attribute functional suitability	52
5.5. Example of valid artifact type implementations	53
5.6. Example of a valid trace link type implementation	54

List of Figures

5.7. Example of a valid trace path type implementation	55
5.8. Example of valid artifact implementation	55
5.9. Example of a valid trace link implementations	56
5.10. Example of a valid trace path implementations	57
5.11. Example of an incomplete traceability implementation that misses artifact type	58
5.12. Example of an incomplete traceability implementation that misses a trace link type	58
5.13. Example of an incomplete traceability implementation that misses a trace path type	59
5.14. Example of an incomplete traceability implementation that misses artifacts	59
5.15. Example of an incomplete traceability implementation that misses trace links	60
5.16. Example of an incomplete traceability implementation that misses trace paths	61
5.17. Example of an inappropriate traceability implementation that contains a superfluous artifact type	61
5.18. Example of an inappropriate traceability implementation that contains a superfluous trace link type	62
5.19. Example of an inappropriate traceability implementation that contains a superfluous trace path type	62
5.20. Example of an inappropriate traceability implementation that contains superfluous artifacts	63
5.21. Example of an inappropriate traceability implementation that contains superfluous trace links	63
5.22. Example of an inappropriate traceability implementation that contains superfluous trace paths	64
5.23. Example of an incorrect traceability implementation that contains a wrong trace link type	65
5.24. Example of an incorrect traceability implementation that contains a wrong trace path type	66
5.25. Example of an inappropriate traceability implementation that contains superfluous trace paths	66
5.26. Example of an incorrect traceability implementation that contains wrong trace links	67
5.27. Example of an inappropriate traceability implementation that contains superfluous trace paths	67
5.28. Example of an inappropriate traceability implementation that contains superfluous trace paths	68

5.29. Dependencies among the traceability implementation data and the derived dependencies among the traceability problems 69

5.30. Overview of the three preparation steps (step 1-3) and the one assessment step (step 4) that are required to perform a traceability assessment . . . 70

5.31. Example of collecting artifacts, trace links, and trace paths from textual specification documents 71

5.32. Example of collecting artifacts, trace links, and trace paths from IBM Rational DOORS 72

5.33. Example of extracting artifact types, trace link types, and trace path types from textual specification documents 73

5.34. Example of mapping the extracted artifact types with the required artifact types of the TIM 75

5.35. Traceability assessment report example 76

6.1. Overview of the PurISTA tool components 80

6.2. Example of a model entity to relational database mapping definition . . 81

6.3. Overview of implemented external interfaces with artifact management tools 83

6.4. Screenshot of the traceability browser that visualizes the traceability requirements derived from the DO-178B standard [DO-178B] and the artifact types of two projects at zoom level zero 85

6.5. Screenshot of the traceability browser shows the currently selected artifact type and all directly related elements 86

6.6. Screenshot of the traceability browser that visualizes the traceability implementation data at artifact level 87

6.7. Example of an assessment report for the traceability problems missing trace path type (\mathcal{M}_P) 87

7.1. Results of study 1: average relevance ratings per traceability problem . . 94

7.2. Roles of the subjects in study 4 103

7.3. Safety domains of the subjects in study 4 104

7.4. Results of study 4: subjects' opinions on the traceability assessment results with respect to indicate compliance problems 105

7.5. Results of study 4: subjects' opinions on the traceability assessment results with respect to indicate safety problems 105

7.6. Results of study 4: certifiers' opinions on the usefulness of the traceability assessment results for their work 106

7.7. Results of study 4: project participants' opinions on the usefulness of the traceability assessment results for their work 106

List of Tables

2.1. Overview of desired traceability qualities and their related goals [Cleland-Huang et al. 2014]	13
7.1. Characteristics of the interviewed subjects, their project, and their company	96
7.2. Results of study 2: assessment results across the 17 studied cases	97
7.3. Characteristics of the formalized safety guidelines	98
7.4. Characteristics of the assessed software projects	99
7.5. Results of study 3: overview of assessment results per project	100
7.6. Results of study 3: comparison of guidelines with respect to required artifact types and required trace paths	101

Bibliography

- Amyot, D. (2003). “Introduction to the User Requirements Notation: learning by example”. en. In: *Computer Networks* 42.3, pp. 285–301.
- ANTLR Team (2015a). *ANother Tool for Language Recognition (ANTLR) [Online]*. <http://www.antlr.org>.
- ANTLR Team (2015b). *Java 1.7 grammar for ANTLR v4 [Online]*. <https://github.com/antlr/grammars-v4/blob/master/java/Java.g4>.
- Anton, A. (1996). “Goal-based requirements analysis”. In: *Proceedings of the Second International Conference on Requirements Engineering*. IEEE Comput. Soc. Press, pp. 136–144.
- Arkley, P. and S. Riddle (2005). “Overcoming the traceability benefit problem”. In: *Proceedings of the 13th IEEE International Conference on Requirements Engineering*. IEEE, pp. 385–389.
- Arkley, P. and S. Riddle (2006). “Tailoring Traceability Information to Business Needs”. In: *Proceedings of the 14th IEEE International Requirements Engineering Conference*. IEEE, pp. 239–244.
- Arkley, P., P. Mason, and S. Riddle (2002). “Position paper: Enabling traceability”. In: *Proceedings of the 1st International Workshop on Traceability in Emerging Forms of Software Engineering, Edinburgh, Scotland (September 2002)*. Citeseer, pp. 61–65.
- Bellamy, J. L., S. E. Bledsoe, and D. E. Traube (2006). “The Current State of Evidence-Based Practice in Social Work: A Review of the Literature and Qualitative Analysis of Expert Interviews”. en. In: *Journal of Evidence-Based Social Work* 3.1, pp. 23–48.
- Bianchi, A., A. Fasolino, and G. Visaggio (2000). “An exploratory case study of the maintenance effectiveness of traceability models”. In: *Proceedings of the 8th International Workshop on Program Comprehension (IWPC)*. IEEE Comput. Soc, pp. 149–158.
- Biernacki, P. and D. Waldorf (1981). “Snowball sampling: Problems and techniques of chain referral sampling”. In: *Sociological methods & research* 10.2, pp. 141–163.
- Boehm, B. W., ed. (1978). *Characteristics of software quality*. TRW series of software technology v. 1. Amsterdam : New York: North-Holland Pub. Co. ; American Elsevier.
- Bohner (1996). “Impact analysis in the software change process: a year 2000 perspective”. In: *Proceedings of the International Conference on Software Maintenance*. IEEE, pp. 42–51.

Bibliography

- Bohner, S. A. (1995). “A Graph Traceability Approach for Software Change Impact Analysis”. PhD thesis. Fairfax, VA, USA: George Mason University.
- Bondy, J. A. and U. S. R. Murty (1976). *Graph theory with applications*. New York: North Holland.
- Breaux, T. D., M. W. Vail, and A. Antón (2006). “Towards regulatory compliance: Extracting rights and obligations to align requirements with regulations”. In: *Proceedings of the 14th IEEE International Conference Requirements Engineering*. IEEE, pp. 49–58.
- Briand, L., Y. Labiche, and G. Soccar (2002). “Automating impact analysis and regression test selection based on UML designs”. In: *Proceedings of the International Conference on Software Maintenance*. IEEE Comput. Soc, pp. 252–261.
- Briand, L., D. Falessi, S. Nejati, M. Sabetzadeh, and T. Yue (2014). “Traceability and SysML design slices to support safety inspections: A controlled experiment”. en. In: *ACM Transactions on Software Engineering and Methodology* 23.1, pp. 1–43.
- Canfora, G. and L. Cerulo (2005). “Impact Analysis by Mining Software and Change Request Repositories”. In: *Proceedings of the 11th IEEE International Symposium Software Metrics*. IEEE, pp. 29–29.
- Casey, V. and F. Mc Caffery (2011). “Med-Trace: traceability assessment method for medical device software development”. In: *Proceedings of the European Systems and Software Process Improvement and Innovation Conference*.
- CENELEC (2011). *EN 50126: Railway applications. The specification and demonstration of reliability, availability, maintainability and safety (RAMS)*. standard.
- Center of Excellence for Software Traceability (COEST) (2015a). *Software Traceability [Online]*. http://coest.org/bok/index.php/Main_Page.
- Center of Excellence for Software Traceability (COEST) (2015b). *What is Traceability? [Online]*. <http://coest.org/index.php/what-is-traceability>.
- Cleland-Huang, J., G. Zemont, and W. Lukasik (2004). “A heterogeneous solution for improving the return on investment of requirements traceability”. In: *Proceedings of the 12th IEEE International Requirements Engineering Conference*. IEEE, pp. 214–223.
- Cleland-Huang, J., B. Berenbach, S. Clark, R. Settini, and E. Romanova (2007). “Best Practices for Automated Traceability”. In: *Computer* 40.6, pp. 27–35.
- Cleland-Huang, J., C. K. Chang, and M. Christensen (2003). “Event-based traceability for managing evolutionary change”. In: *IEEE Transactions on Software Engineering* 29.9, pp. 796–810.
- Cleland-Huang, J., A. Czauderna, M. Gibiec, and J. Emenecker (2010). “A machine learning approach for tracing regulatory codes to product specific requirements”. en. In: *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering*. Vol. 1. ACM Press, p. 155.

- Cleland-Huang, J., O. C. Z. Gotel, J. Huffman Hayes, P. Mäder, and A. Zisman (2014). “Software traceability: trends and future directions”. en. In: *Proceedings of the Future of Software Engineering*. ACM Press, pp. 55–69.
- Cleland-Huang, J., J. H. Hayes, and J. M. Domel (2009). “Model-based traceability”. In: *Proceedings of the ICSE Workshop on Traceability in Emerging Forms of Software Engineering*. IEEE, pp. 6–10.
- Cleland-Huang, J., M. Heimdahl, J. Huffman Hayes, R. Lutz, and P. Maeder (2012). “Trace Queries for Safety Requirements in High Assurance Systems”. In: *Requirements Engineering: Foundation for Software Quality*. Ed. by D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, B. Regnell, and D. Damian. Vol. 7195. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 179–193.
- CMMI Product Team (2006). *CMU/SEI-2006-TR-008: CMMI for Development, Version 1.2*. Tech. rep. Carnegie Mellon University / Software Engineering Institute.
- Comar, C., F. Gasperoni, and J. Ruiz (2009). “Open-DO: an open-source initiative for the development of safety-critical software”. en. In: *Proceedings of the 4th IET International Conference on System Safety*. IET, P3–P3.
- Costello, R. J. and D.-B. Liu (1995). “Metrics for requirements engineering”. en. In: *Journal of Systems and Software* 29.1, pp. 39–63.
- Curtis, S., W. Gesler, G. Smith, and S. Washburn (2000). “Approaches to sampling and case selection in qualitative research: examples in the geography of health”. en. In: *Social Science & Medicine* 50.7-8, pp. 1001–1014.
- Davis, A., S. Overmyer, K. Jordan, J. Caruso, F. Dandashi, A. Dinh, G. Kincaid, G. Ledebor, P. Reynolds, P. Sitaram, A. Ta, and M. Theofanos (1993). “Identifying and measuring quality in a software requirements specification”. In: *Proceedings of the First International Software Metrics Symposium*. IEEE Comput. Soc. Press, pp. 141–152.
- De Lucia, A., F. Fasano, and R. Oliveto (2008). “Traceability management for impact analysis”. In: *Proceedings of the Frontiers of Software Maintenance conference*. IEEE, pp. 21–30.
- Delater, A. and B. Paech (2013). “Tracing Requirements and Source Code during Software Development: An Empirical Study”. In: *Proceedings of the ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*. IEEE, pp. 25–34.
- Deming, W. E. (2000). *Out of the crisis*. eng. 1. MIT Press ed. Cambridge, Mass.: The MIT Press.
- Dick, J. (2002). “Rich traceability”. In: *Proceedings of the 1st international workshop on traceability in emerging forms of software engineering, Edinburgh, Scotland*, pp. 18–23.

- Dijkstra, E. W. (1959). “A note on two problems in connexion with graphs”. en. In: *Numerische Mathematik* 1.1, pp. 269–271.
- Dömges, R. and K. Pohl (1998). “Adapting traceability environments to project-specific needs”. In: *Communications of the ACM* 41.12, pp. 54–62.
- Ebner, G. and H. Kaindl (2002). “Tracing all around in reengineering”. en. In: *IEEE Software* 19.3, pp. 70–77.
- ECSS (2009). *ECSS-E-ST-40C: Space engineering – Software*. standard.
- Egyed, A. (2001). “A scenario-driven approach to traceability”. In: *Proceedings of the 23rd international conference on Software engineering*. IEEE Computer Society, pp. 123–132.
- Farail, P., P. Gauffillet, A. Canals, C. Le Camus, D. Sciamma, P. Michel, X. Crégut, and M. Pantel (2006). “The TOPCASED project: a toolkit in open source for critical aeronautic systems design”. In: *Embedded Real Time Software (ERTS)* 781, pp. 54–59.
- FDA (2002). *General Principles of Software Validation; Final Guidance for Industry and FDA Staff*. standard. Food and Drug Administration.
- Fitzpatrick, J. L., C. A. Christie, and M. M. Mark, eds. (2009). *Evaluation in action: interviews with expert evaluators*. Los Angeles: Sage Publications.
- Gene-Auto Team (2013). *Gene-Auto [Online]*. <http://gforge.enseeiht.fr/projects/geneauto>.
- Git Team (2015). *Git: a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency [Online]*. <http://git-scm.com>.
- Gorden, R. L. (1980). *Interviewing: strategy, techniques, and tactics*. 3d ed. The Dorsey series in sociology. Homewood, Ill. : Georgetown, Ont: Dorsey Press ; Irwin-Dorsey Ltd.
- Gordon, D. G. and T. D. Breaux (2013). “A cross-domain empirical study and legal evaluation of the requirements water marking method”. en. In: *Requirements Engineering* 18.2, pp. 147–173.
- Gotel, O. and A. Finkelstein (1997). “Extended requirements traceability: results of an industrial case study”. In: *Proceedings of the 3rd IEEE International Symposium on Requirements Engineering*. IEEE Comput. Soc. Press, pp. 169–178.
- Gotel, O. and C. Finkelstein (1994). “An analysis of the requirements traceability problem”. In: *Proceedings of 1st International Conference on Requirements Engineering*. IEEE, pp. 94–101.
- Gotel, O., J. Cleland-Huang, J. H. Hayes, A. Zisman, A. Egyed, P. Grünbacher, A. Dekhtyar, G. Antoniol, and J. Maletic (2012a). “The Grand Challenge of Traceability (v1.0)”. en. In: *Software and Systems Traceability*. Ed. by J. Cleland-Huang, O. Gotel, and A. Zisman. London: Springer London, pp. 343–409.
- Gotel, O., J. Cleland-Huang, J. H. Hayes, A. Zisman, A. Egyed, P. Grünbacher, A. Dekhtyar, G. Antoniol, J. Maletic, and P. Mäder (2012b). “Traceability Fundamen-

- tals”. en. In: *Software and Systems Traceability*. Ed. by J. Cleland-Huang, O. Gotel, and A. Zisman. Springer, pp. 3–22.
- Grand View Research (2014). *Embedded System Market Analysis By Product (Hardware, Software), By Application (Automotive, Telecommunication, Healthcare, Industrial, Consumer Electronics, Military & Aerospace) And Segment Forecasts To 2020 [Online]*. <http://www.grandviewresearch.com/industry-analysis/embedded-system-market>.
- Hayes, J. H. and A. Dekhtyar (2005). “Humans in the traceability loop: can’t live with ’em, can’t live without ’em”. en. In: *Proceedings of the 3rd international workshop on Traceability in emerging forms of software engineering*. ACM Press, p. 20.
- Hayes, J. H., A. Dekhtyar, S. K. Sundaram, E. A. Holbrook, S. Vadlamudi, and A. April (2007). “REquirements TRacing On target (RETRO): improving software maintenance through traceability recovery”. en. In: *Innovations in Systems and Software Engineering 3.3*, pp. 193–202.
- Heindl, M. and S. Biffel (2005). “A case study on value-based requirements tracing”. en. In: *Proceedings of the 10th European software engineering conference*. ACM Press, pp. 60–69.
- Heindl, M. and S. Biffel (2006). *Requirements Tracing Strategies for Change Impact Analysis and Re-Testing*. Technical Report. Institute of Software Technology and Interactive Systems Vienna University of Technology.
- Huhns, M. and M. Singh (2005). “Service-oriented computing: key concepts and principles”. en. In: *IEEE Internet Computing 9.1*, pp. 75–81.
- Hull, E., K. Jackson, and J. Dick (2011). *Requirements engineering*. 3rd. Springer.
- IBM (2015a). *DB2 [Online]*. <http://www.ibm.com/software/data/db2/>.
- IBM (2015b). *Rational DOORS [Online]*. <http://www.ibm.com/software/products/en/ratidoor>.
- IEC (2010). *IEC 61508: Functional safety of electrical/electronic/programmable electronic safety-related systems*. Standard. IEC.
- Ingram, C. and S. Riddle (2012). “Cost-Benefits of Traceability”. en. In: *Software and Systems Traceability*. Ed. by J. Cleland-Huang, O. Gotel, and A. Zisman. Springer, pp. 23–42.
- “International Requirements Engineering Conference (RE)” (2015). In:
- “International Symposium on Software and Systems Traceability (SST)” (2015). In:
- ISO (2008). *ISO 9001:2008 – Quality management systems – Requirements*. Tech. rep. ISO.
- ISO (2011a). *ISO 25010:2011 – Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models*. standard. ISO.
- ISO (2011b). *ISO 26262-6:2011 – Road vehicles – Functional safety – Part 6: Product development at the software level*. standard. ISO.

- ISO and IEC (2004). *ISO/IEC 15504:2004 Information technology – Process assessment*. Standard. ISO/IEC.
- ISO, IEC, and IEEE (2011a). *ISO/IEC/IEEE 12207:2008 – Standard for Systems and Software Engineering – Software Life Cycle Processes*. standard. ISO/IEC/IEEE.
- ISO, IEC, and IEEE (2011b). *ISO/IEC/IEEE 29148:2011 – Systems and software engineering – Life cycle processes – Requirements engineering*. standard. ISO/IEC/IEEE.
- Atlassian (2015). *Jira [Online]*. <http://www.atlassian.com/software/jira>.]
- Jönsson, P. and M. Lindvall (2005). “Impact Analysis”. en. In: *Engineering and Managing Software Requirements*. Ed. by A. Aurum and C. Wohlin. Berlin/Heidelberg: Springer-Verlag, pp. 117–142.
- Kelly, T. P. (1999). *Arguing safety-a systematic approach to managing safety cases*. University of York.
- Kirova, V., N. Kirby, D. Kothari, and G. Childress (2008). “Effective requirements traceability: Models, tools, and practices”. en. In: *Bell Labs Technical Journal* 12.4, pp. 143–157.
- Kitchenham, B. A. and S. L. Pfleeger (2008). “Personal Opinion Surveys”. en. In: *Guide to Advanced Empirical Software Engineering*. Ed. by F. Shull, J. Singer, and D. I. K. Sjøberg. London: Springer London, pp. 63–92.
- Kornecki, A. J. and J. Zalewski (2005). “Experimental evaluation of software development tools for safety-critical real-time systems”. en. In: *Innovations in Systems and Software Engineering* 1.2, pp. 176–188.
- Kroll, P. and P. Kruchten (2003). *The rational unified process made easy: a practitioner’s guide to the RUP*. Addison-Wesley object technology series. Boston: Addison-Wesley.
- Leffingwell, D. (1997). “Calculating your return on investment from more effective requirements management”. In: *American Programmer* 10.4, pp. 13–16.
- LibGit2Sharp Team (2015). *LibGit2Sharp [Online]*. <http://github.com/libgit2/libgit2sharp>.
- Lindvall, M. and K. Sandahl (1996). “Practical implications of traceability”. In: *Software Practice and Experience* 26.10, pp. 1161–1180.
- Lormans, M. and A. van Deursen (2005). “Reconstructing requirements coverage views from design and test using traceability recovery via LSI”. en. In: *Proceedings of the 3rd international workshop on Traceability in emerging forms of software engineering*. ACM Press, p. 37.
- Mäder, P. (2010). *Rule-based maintenance of post-requirements traceability*. eng. Münster: MV-Verl.
- Mäder, P., O. Gotel, and I. Philippow (2008). “Rule-Based Maintenance of Post-Requirements Traceability Relations”. In: *Proceedings of the 16th IEEE International Requirements Engineering Conference*. IEEE, pp. 23–32.
- Mäder, P., O. Gotel, and I. Philippow (2009a). “Getting back to basics: Promoting the use of a traceability information model in practice”. In: *Proceedings of the ICSE*

- Workshop on Traceability in Emerging Forms of Software Engineering*. IEEE, pp. 21–25.
- Mäder, P., O. Gotel, and I. Philippow (2009b). “Motivation Matters in the Traceability Trenches”. In: *Proceedings of the 17th IEEE International Requirements Engineering Conference*. IEEE, pp. 143–148.
- Mäder, P., P. L. Jones, Y. Zhang, and J. Cleland-Huang (2013). “Strategic Traceability for Safety-Critical Projects”. In: *IEEE Software* 30.3, pp. 58–66.
- Mäder, P., I. Philippow, and M. Riebisch (2007). “A Traceability Link Model for the Unified Process”. In: *Proceedings of the 8th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing*. IEEE, pp. 700–705.
- Maletic, J. I. and M. L. Collard (2009). “TQL: A query language to support traceability”. In: *Proceedings of the ICSE Workshop on Traceability in Emerging Forms of Software Engineering*. IEEE, pp. 16–20.
- Marcus, A. and J. Maletic (2003). “Recovering documentation-to-source-code traceability links using latent semantic indexing”. In: *Proceedings of the 25th International Conference on Software Engineering*. IEEE, pp. 125–135.
- McCall, J. A., P. K. Richards, and G. F. Walters (1977). *Factors in software quality. Volume 1. Concepts and definitions of software quality*. Technical Report. DTIC Document.
- Mercurial Team (2015). *Mercurial [Online]*. <https://www.mercurial-scm.org/>.
- Merilinna, J. and J. Pärssinen (2010). “Verification and validation in the context of domain-specific modelling”. en. In: *Proceedings of the 10th Workshop on Domain-Specific Modeling*. ACM Press, p. 1.
- Microsoft (2015a). *Microsoft SQL Server [Online]*. <https://www.microsoft.com/en-us/server-cloud/products/sql-server/default.aspx>.
- Microsoft (2015b). *Microsoft Word [Online]*. <http://products.office.com/word>.
- Middleton, P., P. Kjeldsen, and J. Tully (2013). *Gartner Forecast: The Internet of Things, Worldwide [Online]*. <https://www.gartner.com/doc/2625419/forecast-internet-things-worldwide>.
- Mirakhorli, M. and J. Cleland-Huang (2011). “Using tactic traceability information models to reduce the risk of architectural degradation during system maintenance”. In: *Proceedings of the 27th IEEE International Conference on Software Maintenance*. IEEE, pp. 123–132.
- Nair, S., J. L. de la Vara, and S. Sen (2013). “A review of traceability research at the requirements engineering conference RE@21”. In: *Proceedings of the 21st IEEE International Requirements Engineering Conference (RE)*. IEEE, pp. 222–229.
- Nejati, S., M. Sabetzadeh, D. Falessi, L. Briand, and T. Coq (2012). “A SysML-based approach to traceability management and design slicing in support of safety certifi-

- ation: Framework, tool support, and case studies”. en. In: *Information and Software Technology* 54.6, pp. 569–590.
- NHibernate (2015). *NHibernate [Online]*. <http://nhibernate.info>.
- OASIS (2015). *Open Services for Lifecycle Collaboration Core Specifications Version 3.0 [Online]*. <http://open-services.net/wiki/core/Specification-3.0>.
- Omoronyia, I., G. Sindre, and T. Stålhane (2011). “Exploring a Bayesian and linear approach to requirements traceability”. en. In: *Information and Software Technology* 53.8, pp. 851–871.
- Apache (2015a). *Open Office [Online]*. <http://www.openoffice.org>.
- Oracle (2015a). *Java [Online]*. <http://java.com>.
- Oracle (2015b). *MySQL [Online]*. <http://www.mysql.com>.
- Oracle (2015c). *Oracle Database [Online]*. <https://www.oracle.com/database>.
- Panesar-Walawege, R. K., M. Sabetzadeh, L. Briand, and T. Coq (2010). “Characterizing the Chain of Evidence for Software Safety Cases: A Conceptual Model Based on the IEC 61508 Standard”. In: *Proceedings of the Third International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, pp. 335–344.
- Pfleeger, S. and S. Bohner (1990). “A framework for software maintenance metrics”. In: *Proceedings of the Conference on Software Maintenance*. IEEE, pp. 320–327.
- Pinheiro, F. and J. Goguen (1996). “An object-oriented tool for tracing requirements”. In: *Proceedings of the Second International Conference on Requirements Engineering*. IEEE Comput. Soc. Press, p. 219.
- Pinheiro, F. A. C. (2004). “Requirements Traceability”. In: *Perspectives on Software Requirements*. Ed. by J. C. S. Prado Leite and J. H. Doorn. Springer, pp. 91–113.
- Pohl, K. (1996). “PRO-ART: enabling requirements pre-traceability”. In: *Proceedings of the 2nd International Conference on Requirements Engineering*. IEEE, pp. 76–84.
- Pohl, K. (2010). *Requirements engineering: fundamentals, principles, and techniques*. Springer.
- Pohl, K., K. Weidenhaupt, R. Dömges, P. Haumer, M. Jarke, and R. Klamma (1999). “PRIME—toward process-integrated modeling environments: 1”. In: *ACM Transactions on Software Engineering and Methodology* 8.4, pp. 343–410.
- Jonathan de Halleux (2015). *QuickGraph: Graph Data Structures And Algorithms for .NET [Online]*. <http://quickgraph.codeplex.com>.
- Ramesh, B. and M. Edwards (1992). “Issues in the development of a requirements traceability model”. In: *Proceedings of IEEE International Symposium on Requirements Engineering*. IEEE Comput. Soc. Press, pp. 256–259.
- Ramesh, B. and M. Jarke (2001). “Toward reference models for requirements traceability”. In: *IEEE Transactions on Software Engineering* 27.1, pp. 58–93.
- Ramesh, B., T. Powers, C. Stubbs, and M. Edwards (1995). “Implementing requirements traceability: a case study”. In: *Proceedings of the 2nd IEEE International Symposium on Requirements Engineering*. IEEE, pp. 89–95.

- Ramesh, B. (1997). “Representing and reasoning with traceability in model life cycle management”. English. In: *Annals of Operations Research* 75, pp. 123–145.
- Ramesh, B. (1998). “Factors influencing requirements traceability practice”. In: *Communications of the ACM* 41.12, pp. 37–44.
- RAMI (2015). *Rate Adjustment by Managing Inflows (RAMI) [Online]*. <http://www.chris-edwards.org/340>.
- Regan, G., M. Biro, F. Mc Caffery, K. Mc Daid, and D. Flood (2014). “A Traceability Process Assessment Model for the Medical Device Domain”. In: *Systems, Software and Services Process Improvement*. Vol. 425. Springer, pp. 206–216.
- Regan, G., F. McCaffery, K. McDaid, and D. Flood (2012). “The Barriers to Traceability and their Potential Solutions: Towards a Reference Framework”. In: *Proceedings of the 38th EUROMICRO Conference on Software Engineering and Advanced Applications*. IEEE, pp. 319–322.
- Rempel, P. and P. Mäder (2015a). “A quality model for the systematic assessment of requirements traceability”. In: *Proceedings of the 23rd IEEE International Requirements Engineering Conference*. IEEE, pp. 176–185.
- Rempel, P. and P. Mäder (2015b). “Estimating the Implementation Risk of Requirements in Agile Software Development Projects with Traceability Metrics”. In: *Requirements Engineering: Foundation for Software Quality*. Ed. by S. A. Fricker and K. Schneider. Vol. 9013. Springer, pp. 81–97.
- Rempel, P., P. Mäder, and T. Kuschke (2013). “An empirical study on project-specific traceability strategies”. In: *Proceedings of the 21st IEEE International Requirements Engineering Conference*. IEEE, pp. 195–204.
- Rempel, P., P. Mäder, T. Kuschke, and J. Cleland-Huang (2014). “Mind the gap: assessing the conformance of software traceability to relevant guidelines”. en. In: *Proceedings of the 36th International Conference on Software Engineering ICSE*. ACM Press, pp. 943–954.
- Ridderhof, W., H.-G. Gross, and H. Doerr (2007). “Establishing Evidence for Safety Cases in Automotive Systems – A Case Study”. In: *Computer Safety, Reliability, and Security*. Ed. by D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, F. Saglietti, and N. Oster. Vol. 4680. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 1–13.
- Rierson, L. (2013). *Developing Safety-Critical Software: A Practical Guide for Aviation Software and DO-178C Compliance*. CRC Press.
- RTCA (2000). *DO-178B: Software Considerations in Airborne Systems and Equipment Certification*. Guideline. RTCA.
- RTCA (2011). *DO-178C: Software Considerations in Airborne Systems and Equipment Certification*. Guideline. RTCA.

- Runeson, P. and M. Höst (2009). “Guidelines for conducting and reporting case study research in software engineering”. en. In: *Empirical Software Engineering* 14.2, pp. 131–164.
- Schreier, M. (2014). “Qualitative content analysis”. In: *The SAGE Handbook of Qualitative Data Analysis*, pp. 170–183.
- Schwarz, H., J. Ebert, and A. Winter (2010). “Graph-based traceability: a comprehensive approach”. en. In: *Software & Systems Modeling* 9.4, pp. 473–492.
- Smith, J. (2009). “PATTERNS-WPF Apps With The Model-View-ViewModel Design Pattern”. In: *MSDN magazine*, p. 72.
- SQLite (2015). *SQLite [Online]*. <http://www.sqlite.org>.
- Stevenson, A. (2010). *Oxford dictionary of English*. Oxford University Press.
- Storey, N. (2010). *Safety-critical computer systems*. eng. Nachdr. Harlow: Prentice Hall.
- Apache (2015b). *Subversion [Online]*. <http://subversion.apache.org>.
- Thompson, S. K. (2012). *Sampling*. 3rd edition. Wiley Series in Probability and Statistics. Hoboken, NJ, USA: John Wiley & Sons, Inc.
- TOPCASE-REQ (2015). *TOPCASE-REQ [Online]*. <http://gforge.enseeiht.fr/projects/topcasedreq>.
- TOPCASE-SAM (2015). *TOPCASE-SAM [Online]*. <http://gforge.enseeiht.fr/projects/topcasedsam>.
- Vanhooff, B., S. Van Baelen, W. Joosen, and Y. Berbers (2007). “Traceability as input for model transformations”. In: *Proceedings of the ECMDA Traceability Workshop*. Citeseer, pp. 37–46.
- VERBI (2015). *MAXQDA - the art of data analysis [Online]*. <http://www.maxqda.com>.
- von Knethen, A. (2002). “Change-oriented requirements traceability. Support for evolution of embedded systems”. In: *Proceedings of the International Conference on Software Maintenance*. IEEE Comput. Soc, pp. 482–485.
- von Knethen, A., B. Paech, F. Kiedaisch, and F. Houdek (2002). “Systematic requirements recycling through abstraction and traceability”. In: *Proceedings of the IEEE Joint International Conference on Requirements Engineering*. IEEE, pp. 273–281.
- von Knethen, A. and B. Paech (2002). “A survey on tracing approaches in Practice and Research”. In: 095.01/E.
- Watkins, R. and M. Neal (1994). “Why and how of requirements tracing”. In: *IEEE Software* 11.4, pp. 104–106.
- Wieringa, R. (1995). *An Introduction to Requirements Traceability*. Technical Report IR-389. Amsterdam, the Netherlands: Free University, Faculty of Mathematics and Computer Science.
- Winkler, S. and J. von Pilgrim (2010). “A survey of traceability in requirements engineering and model-driven development”. en. In: *Software & Systems Modeling* 9.4, pp. 529–565.

- Wohlin, C., P. Runeson, M. Höst, M. Ohlsson, B. Regnell, and A. Wesslen (2012). *Experimentation in software engineering*. New York: Springer.
- Yin, R. K. (2009). *Case study research: design and methods*. 4th. Applied social research methods. Sage Publications.
- Zeller, M., K. Höfig, and M. Rothfelder (2014). “Towards a Cross-Domain Software Safety Assurance Process for Embedded Systems”. In: *Computer Safety, Reliability, and Security*. Ed. by A. Bondavalli, A. Ceccarelli, and F. Ortmeier. Vol. 8696. Cham: Springer International Publishing, pp. 396–400.

A. Evaluation Material of Study 1


Questionnaire for a Classification
of Structural Software Traceability Problems



Introduction

We developed a classification of all possible **Structural Traceability Problems**, which may occur in a software development project, and which can be used to systematically assess the quality of a projects' software traceability. **Semantical Traceability Problems** such as incorrect trace link between artifacts are **out-of-scope** of this classification.

In this questionnaire, we ask for your opinion on our classification based on your experience.

Questions, we would like you to answer are marked with a  symbol.

The questionnaire consists of four parts:


- Part 1. We ask preliminary questions about your software traceability experience.
- Part 2. To make you familiar with our preliminary assumptions from which we derived the classification, we provide **definitions** of the relevant traceability elements, traceability actors, and sets of elements.
- Part 3. We present our developed classification and ask for your opinion. Thereby, for every traceability element, we define a **Quality Gate** describing its acceptable state and **Traceability Problems** describing unacceptable deviations from this acceptable state.
- Part 4. We ask for your opinion whether or not the developed classification is exhaustive.


With the scale: MINOR MAJOR you rate the **relevance** of a **Quality Gate**.


With the scale: NONCRITICAL CRITICAL you rate the **criticality** of a **Problem**.

Please check one circle of each scale as exemplified above.

Part 1: Preliminary Questions

 Q₁: Please indicate your practical software traceability experience in years:

 Q₂: Have you ever **participated** in a project that captured traceability information? Yes No

 Q₃: Have you ever **assessed** the quality of captured traceability information in a project? Yes No

Part 2: Software Traceability Definitions

The following two figures are supposed to make you familiar with our preliminary assumptions from which we derived our classification of structural traceability problems.

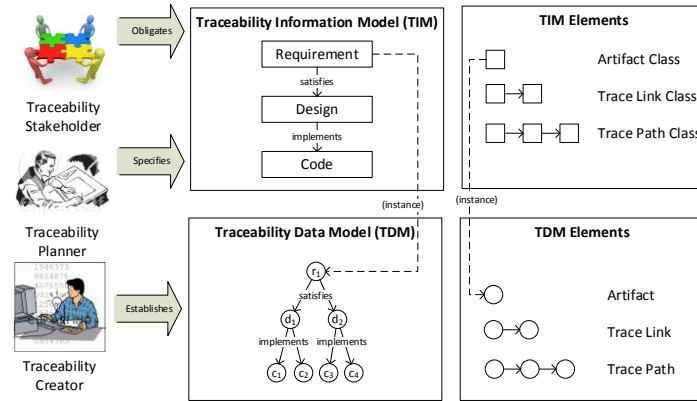


Figure 1 – The figure shows the interplay of traceability elements and traceability actors in a software project. As depicted in the upper part, a Traceability Planner specifies traceability (modeled as TIM) in order to address the projects' traceability obligations, which are stated by one or many Traceability Stakeholders (e.g. client, organization, regulation, etc.). As depicted in the lower part, a Traceability Creator establishes traceability (modeled as TDM) in order to satisfy the Traceability Planners' specified traceability.

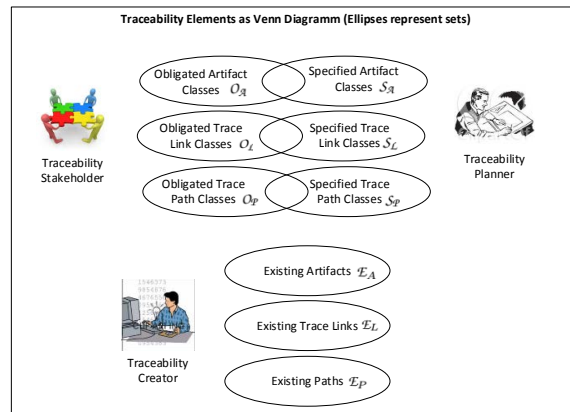
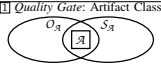
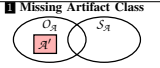
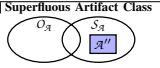

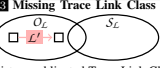
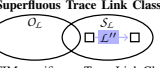
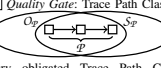
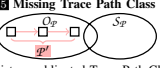

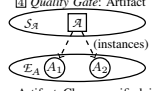
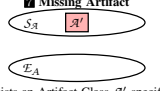
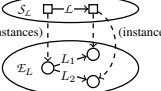
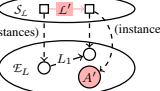
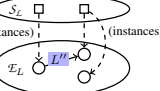
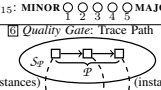

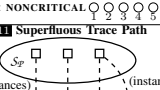



Figure 2 – Depending on the actual actor, traceability elements can be distinguished as obligated, specified, and existing. This figure depicts all sets of traceability elements as Venn diagram, which can be derived from all possible traceability elements as depicted in Figure 1 and the actor specific distinction of traceability elements.

Part 3: Traceability Problem Classification

		Traceability Quality Gates 1 - 6		Traceability Problems 1 - 11	
Traceability Information Model (TIM)	Artifact Class (A)	<p>1 Quality Gate: Artifact Class</p>  <p>Every obligated Artifact Class should be specified in the TIM and every Artifact Class specified in the TIM should be obligated.</p> <p>Q4: MINOR <input type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3 <input type="radio"/> 4 <input type="radio"/> 5 MAJOR</p>	<p>Missing Traceability Elements</p> <p>4 Missing Artifact Class</p>  <p>It exists an obligated Artifact Class A', which is not specified within the TIM.</p> <p>Q5: NONCRITICAL <input type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3 <input type="radio"/> 4 <input type="radio"/> 5 CRITICAL</p>	<p>Superfluous Traceability Elements</p> <p>2 Superfluous Artifact Class</p>  <p>The TIM specifies an Artifact Class A'', which is not obligated.</p> <p>Q6: NONCRITICAL <input type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3 <input type="radio"/> 4 <input type="radio"/> 5 CRITICAL</p>	
	Trace Link Class (L)	<p>2 Quality Gate: Trace Link Class</p>  <p>Every obligated Trace Link Class should be specified in the TIM and every Trace Link Class specified in the TIM should be obligated.</p> <p>Q7: MINOR <input type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3 <input type="radio"/> 4 <input type="radio"/> 5 MAJOR</p>	<p>5 Missing Trace Link Class</p>  <p>It exists an obligated Trace Link Class L', which is not specified within the TIM.</p> <p>Q8: NONCRITICAL <input type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3 <input type="radio"/> 4 <input type="radio"/> 5 CRITICAL</p>	<p>3 Superfluous Trace Link Class</p>  <p>The TIM specifies an Trace Link Class L'', which is not obligated.</p> <p>Q9: NONCRITICAL <input type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3 <input type="radio"/> 4 <input type="radio"/> 5 CRITICAL</p>	
	Trace Path Class (P)	<p>3 Quality Gate: Trace Path Class</p>  <p>Every obligated Trace Path Class should be specified in the TIM and every Trace Path Class specified in the TIM should be obligated.</p> <p>Q10: MINOR <input type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3 <input type="radio"/> 4 <input type="radio"/> 5 MAJOR</p>	<p>6 Missing Trace Path Class</p>  <p>It exists an obligated Trace Path Class P', which is not specified within the TIM.</p> <p>Q11: NONCRITICAL <input type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3 <input type="radio"/> 4 <input type="radio"/> 5 CRITICAL</p>	<p>4 Superfluous Trace Path Class</p>  <p>The TIM specifies an Trace Path Class P'', which is not obligated.</p> <p>Q12: NONCRITICAL <input type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3 <input type="radio"/> 4 <input type="radio"/> 5 CRITICAL</p>	
Traceability Data Model (TDM)	Artifact (A)	<p>4 Quality Gate: Artifact</p>  <p>Every Artifact Class specified in the TIM should be instantiated in the TDM as one or many Artifacts.</p> <p>Q13: MINOR <input type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3 <input type="radio"/> 4 <input type="radio"/> 5 MAJOR</p>		<p>7 Missing Artifact</p>  <p>It exists an Artifact Class A' specified in the TIM for which no Artifact instances exists in the TDM.</p> <p>Q14: NONCRITICAL <input type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3 <input type="radio"/> 4 <input type="radio"/> 5 CRITICAL</p>	
	Trace Link (L)	<p>5 Quality Gate: Trace Link</p>  <p>Every Artifact in the TDM, which is an instance of a source or target of a Trace Link Class L, should have one or many Trace Links that instantiate the L.</p> <p>Q15: MINOR <input type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3 <input type="radio"/> 4 <input type="radio"/> 5 MAJOR</p>	<p>8 Missing Trace Link</p>  <p>It exists an Artifact A' in the TDM which is an instance of a source or target of a Trace Link Class L', but has no Trace Link that satisfies L'.</p> <p>Q16: NONCRITICAL <input type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3 <input type="radio"/> 4 <input type="radio"/> 5 CRITICAL</p>	<p>9 Superfluous Trace Link</p>  <p>It exists a Trace Link L'' in the TDM, which does not satisfy any Trace Link Class of the TIM.</p> <p>Q17: NONCRITICAL <input type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3 <input type="radio"/> 4 <input type="radio"/> 5 CRITICAL</p>	
	Trace Path (P)	<p>6 Quality Gate: Trace Path</p>  <p>Every Artifact in the TDM, which is an instance of a source or target of a Trace Path Class P, should have one or many Trace Paths that instantiate the P.</p> <p>Q18: MINOR <input type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3 <input type="radio"/> 4 <input type="radio"/> 5 MAJOR</p>	<p>10 Missing Trace Path</p>  <p>It exists an Artifact A' in the TDM which is an instance of a source or target of a Trace Path Class P', but has no Trace Path that satisfies P'.</p> <p>Q19: NONCRITICAL <input type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3 <input type="radio"/> 4 <input type="radio"/> 5 CRITICAL</p>	<p>11 Superfluous Trace Path</p>  <p>It exists a Trace Path P'' in the TDM, which does not satisfy any Trace Path Class of the TIM.</p> <p>Q20: NONCRITICAL <input type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3 <input type="radio"/> 4 <input type="radio"/> 5 CRITICAL</p>	

Part 4: Missing Problems

 Q₂₁: Are you aware of any additional **Structural Traceability Problem**, which is not covered by our problem current classification? If yes, could you briefly describe which problems?


B. Evaluation Material of Study 2

Questionnaire

In this questionnaire we ask for your opinion on a technique we developed to automatically assess the traceability compliance of software development data with functional safety standards. The questionnaire is structured as follows:

1. We provide an **Introduction** to relevant terms (e.g. traceability, functional safety, safety standards) and explain our traceability compliance assessment technique.
2. We ask **Preliminary Questions** about your software engineering background.
3. We ask for **Your Opinion** on our traceability compliance assessment technique based on your practical experience.

Additional facts:

- Completing this questionnaire will take approximately **30 minutes**.
- Please answer **all questions** that are marked with .
- This questionnaire is **anonymous** and the results are used for research only.
- Target audience: people who are/were concerned (in)directly with the development of software-based systems that need to achieve functional safety.
- All participants of this survey will receive a **detailed report** on the results and are welcome to run the compliance assessment on their own software projects data.

Contact informations:

✉ Technische Universität Ilmenau
Patrick Rempel
Software Systems / Process Informatics
Helmholtzplatz 5
98693 Ilmenau

@ patrick.rempel@tu-ilmenau.de

☎ +49 3677 / 69-4182

Introduction

Many **software safety standards**¹ prescribe **traceability**² to demonstrate that a rigorous software development process has been followed. Traceability *supports* the achievement of **Functional Safety**³ of software-based systems, because it ensures that:

- all **potential hazards, risks and regulations** are *addressed* by requirements,
- all **requirements** are *implemented* and *verified*,
- no **unintended functionality** is *implemented* but only required functionality.

Increasingly, certification to software safety standards is desired by end-users and encouraged by regulatory authorities to minimize the risk of use.



It is the **certifier's** responsibility during certification to *check* whether or not a developed **software product** *complies* with a **standard**. In terms of traceability, the **certifier** needs to check whether or not the software development data (e.g. requirements, design document, source code, test cases) are traceable as prescribed by a standard.



Software engineers, developing safety-critical software, have the responsibility to *develop* **software products** that *comply* with a **standard**. Thus, **software engineers** need to ensure that all software development data (e.g. requirements, design, source code, test cases) are traceable as prescribed by a standard.



We have developed an **assessment tool** that *automatically checks* **traceability compliance** of software development data to safety standards. The tool's main goal is to *support* **software engineers** and **certifiers**.

The traceability compliance assessment tool works as follows:

- We defined **5 problem classes**. We *claim* that the existence of such problems in software development data *indicate* the **non-compliance** of traceability.
- The tool *searches* for **problem instances** within the software development data.
- The tool *generates* **assessment reports** for every problem class.

In this questionnaire, we provide **illustrating examples**, which are *based* on the **DO-178B** standard for software within aircrafts⁴ and on the industrial development project **Gene-Auto**, which has the explicit project goal to comply with the DO-178B standard.










¹A **software safety standard** specifies requirements for software development that need to be satisfied to ensure that the developed software is safe for use.

²**Traceability** is the ability to trace the origin, the evolution, and the result of any software development data (e.g. requirements, design, source code), even after the development has been completed.



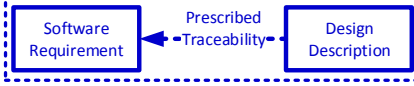
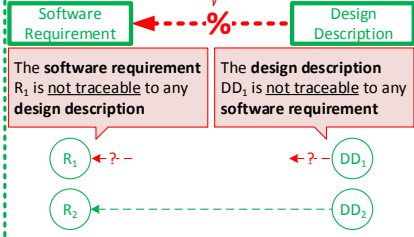




³The objective of **Functional Safety** is freedom from unacceptable risk of physical injury or of damage to the health of people either directly or indirectly through damage to property or to the environment.

⁴**DO-178B**: Software Considerations in Airborne Systems and Equipment Certification.

Preliminary Questions

	Please rate your practical experience in software development projects. years (Please round to full years)												
	What is your primary role in software engineering? (Please select one) <table border="0"> <tr> <td>Developer</td> <td>Tester</td> <td>Consultant</td> <td>Safety Engineer</td> </tr> <tr> <td>Manager</td> <td>Analyst</td> <td>Administrator</td> <td>Certification Engineer</td> </tr> <tr> <td>Architect</td> <td>Auditor</td> <td>Trainer</td> <td></td> </tr> </table>	Developer	Tester	Consultant	Safety Engineer	Manager	Analyst	Administrator	Certification Engineer	Architect	Auditor	Trainer	
Developer	Tester	Consultant	Safety Engineer										
Manager	Analyst	Administrator	Certification Engineer										
Architect	Auditor	Trainer											
	Are / were you concerned with additional role(s) in software engineering? (Please select all that apply) <table border="0"> <tr> <td>Developer</td> <td>Tester</td> <td>Consultant</td> <td>Safety Engineer</td> </tr> <tr> <td>Manager</td> <td>Analyst</td> <td>Administrator</td> <td>Certification Engineer</td> </tr> <tr> <td>Architect</td> <td>Auditor</td> <td>Trainer</td> <td></td> </tr> </table>	Developer	Tester	Consultant	Safety Engineer	Manager	Analyst	Administrator	Certification Engineer	Architect	Auditor	Trainer	
Developer	Tester	Consultant	Safety Engineer										
Manager	Analyst	Administrator	Certification Engineer										
Architect	Auditor	Trainer											
	In which domain(s) are/were you working? (Please select all that apply) <table border="0"> <tr> <td>Aviation</td> <td>Automotive</td> <td>Energy</td> <td>Railway</td> </tr> <tr> <td>Space</td> <td>Medical</td> <td>Agriculture</td> <td>Industrial Automation</td> </tr> <tr> <td>Military</td> <td>Maritime</td> <td>Mining</td> <td></td> </tr> </table>	Aviation	Automotive	Energy	Railway	Space	Medical	Agriculture	Industrial Automation	Military	Maritime	Mining	
Aviation	Automotive	Energy	Railway										
Space	Medical	Agriculture	Industrial Automation										
Military	Maritime	Mining											
	Are you familiar with functional safety of software-based systems? (Please select one) <table border="0"> <tr> <td>Not familiar</td> <td>I know the objectives</td> <td>≤ 1 year experience</td> <td>≥ 2 years experience</td> <td>Expert</td> </tr> </table>	Not familiar	I know the objectives	≤ 1 year experience	≥ 2 years experience	Expert							
Not familiar	I know the objectives	≤ 1 year experience	≥ 2 years experience	Expert									
	Have you participated in a project that was aiming to ensure functional safety? <table border="0"> <tr> <td>Yes</td> <td>No</td> <td>(Please select one)</td> </tr> </table>	Yes	No	(Please select one)									
Yes	No	(Please select one)											
	Have you participated in a project that was certified for functional safety? <table border="0"> <tr> <td>Yes</td> <td>No</td> <td>(Please select one)</td> </tr> </table>	Yes	No	(Please select one)									
Yes	No	(Please select one)											
	Have you certified the functional safety of a software project? <table border="0"> <tr> <td>Yes</td> <td>No</td> <td>(Please select one)</td> </tr> </table>	Yes	No	(Please select one)									
Yes	No	(Please select one)											
	Which of the following functional safety standards are/were relevant for your current or previous work? (Please select all that apply) <table border="0"> <tr> <td>IEC 61508</td> <td>IEC 61513</td> <td>DO-178 B/C</td> <td>IEC 62304</td> </tr> <tr> <td>ISO 26262</td> <td>EN 50128</td> <td>ISO 25119</td> <td></td> </tr> </table>	IEC 61508	IEC 61513	DO-178 B/C	IEC 62304	ISO 26262	EN 50128	ISO 25119					
IEC 61508	IEC 61513	DO-178 B/C	IEC 62304										
ISO 26262	EN 50128	ISO 25119											

Your Opinion - 1st Problem Class

 	<p>Traceability between two types of software development data is <i>incomplete</i>.</p> <p>Illustrating example:</p> <div style="border: 1px dashed blue; padding: 5px; margin-bottom: 10px;"> <p><i>Standard</i></p>  </div> <p>The DO-178B standard prescribes: “Design descriptions developed during the design process should be traceable to software requirements”.</p> <hr/> <div style="border: 1px dashed green; padding: 5px;"> <p><i>Project</i></p> <p>Traceability between software requirements and design descriptions is <i>incomplete</i></p>  <p>The software requirement R_1 is not traceable to any design description</p> <p>The design description DD_1 is not traceable to any software requirement</p> </div> <p>Situation within the project: Software requirements and design descriptions are available. Traceability between software requirements and design descriptions is available. Though, the traceability is <i>incomplete</i>, because the software requirement R_1 is not traceable to any design description and the design description DD_1 is not traceable to any software requirement.</p>
	<p>We claim that <i>incomplete traceability</i> indicates non-compliance of traceability with a standard.</p> <p>What is your opinion based on your practical experience? (Please select one)</p> <p>Strongly Agree Agree Disagree Strongly Disagree Don't know</p>
	<p>Can you briefly explain why you agree or disagree?</p>
	<p>We claim that <i>incomplete traceability</i> indicates safety risks.</p> <p>What is your opinion based on your practical experience? (Please select one)</p> <p>Strongly Agree Agree Disagree Strongly Disagree Don't know</p>
	<p>Can you briefly explain why you agree or disagree?</p>

Exemplary assessment report for the 1st problem class:



```

Traceability Compliance Report
***** Incomplete Traceability *****
Assessed Safety Standard: DO-178-B
Assessed Software Project: GeneAuto

Traceability required in DO-178-B:
>>> System Requirements Data <=> Software Requirements Data <<<
>>> System Requirements Data <=> Software Requirements Data <=> Low-level Requirements Data <<<
>>> Software Requirements Data <=> Software Architecture <<<
>>> Software Requirements Data <=> Low-level Requirements Data <<<
>>> Software Requirements Data <=> Low-level Requirements Data <=> Source Code <<<
>>> Software Requirements Data <=> Software Verification Case <<<
>>> Source Code <=> Low-level Requirements Data <<<
>>> Source Code <=> Low-level Requirements Data <=> Software Verification Case <<<

Incomplete traceability in GeneAuto:
>>> Software Requirements Data <=> INCOMPLETE=> Low-level Requirements Data <<<

Incompleteness summary for development data of type Software Requirements Data:
>>> Total number of Software Requirements Data: 69 <<<
>>> Number of Software Requirements Data with *MISSING* traceability: 8 <<<
>>> Percentage of Software Requirements Data with *MISSING* traceability: 12 % <<<

Detailed report of the 8 Software Requirements Data without traceability:
>>> R-0-010 [Software Requirements Data] <=> MISSING=> ??? [Low-level Requirements Data] <<<
... [For simplicity reasons, this is an excerpt of the complete list only.]

Incompleteness summary for development data of type Low-level Requirements Data:
>>> Total number of Low-level Requirements Data: 34 <<<
>>> Number of Low-level Requirements Data with *MISSING* traceability: 207 <<<
>>> Percentage of Low-level Requirements Data with *MISSING* traceability: 66 % <<<

Detailed report of the 207 Low-level Requirements Data without traceability:
>>> GR-IG-0L06 [Low-level Requirements Data] <=> MISSING=> ??? [Software Requirements Data] <<<
... [For simplicity reasons, this is an excerpt of the complete list only.]

***** Incomplete Traceability *****
    
```



Is the reported information helpful for a **certifier** to check the compliance of **GeneAuto's** traceability with the **DO-178B** standard?
(Please select one answer)

Strongly Agree Agree Disagree Strongly Disagree Don't know



Can you briefly explain why you agree or disagree?



Is the reported information helpful for a **project participant** to ensure the compliance of **GeneAuto's** traceability with the **DO-178B** standard?
(Please select one answer)

Strongly Agree Agree Disagree Strongly Disagree Don't know



Can you briefly explain why you agree or disagree?

Your Opinion - 2nd Problem Class

	<p>Traceability between two types of software development data is completely missing.</p>
	<p>Illustrating example:</p> <div style="display: flex; justify-content: space-between;"> <div style="border: 1px dashed blue; padding: 5px; width: 45%;"> <p><i>Standard</i></p> </div> <div style="width: 45%;"> <p>The DO-178B standard prescribes: “The verification process provides traceability between software requirements and test cases”.</p> </div> </div> <hr/> <div style="display: flex; justify-content: space-between;"> <div style="border: 1px dashed green; padding: 5px; width: 45%;"> <p><i>Project</i></p> <p style="text-align: center; color: red;">Traceability between software requirements and test cases is completely missing</p> <div style="display: flex; justify-content: space-around;"> <div style="border: 1px solid green; padding: 5px; width: 45%;"> <p>Software Requirement</p> </div> <div style="font-size: 2em; color: red; font-weight: bold;">?</div> <div style="border: 1px solid green; padding: 5px; width: 45%;"> <p>Test Case</p> </div> </div> <div style="display: flex; justify-content: space-between; margin-top: 10px;"> <div style="border: 1px solid red; padding: 5px; width: 45%;"> <p>All software requirements (R₁, R₂) are not traceable to any test case</p> <div style="display: flex; justify-content: space-around;"> <div style="border: 1px solid red; border-radius: 50%; padding: 2px;">R₁</div> <div style="color: red;">← ? -</div> </div> <div style="border: 1px solid red; border-radius: 50%; padding: 2px;">R₂</div> <div style="color: red;">← ? -</div> </div> <div style="border: 1px solid red; padding: 5px; width: 45%;"> <p>All test cases (TC₁, TC₂) are not traceable to any software requirement</p> <div style="display: flex; justify-content: space-around;"> <div style="color: red;">← ? -</div> <div style="border: 1px solid red; border-radius: 50%; padding: 2px;">TC₁</div> </div> <div style="display: flex; justify-content: space-around;"> <div style="color: red;">← ? -</div> <div style="border: 1px solid red; border-radius: 50%; padding: 2px;">TC₂</div> </div> </div> </div> </div> </div> <div style="width: 45%;"> <p>Situation within the project: Software requirements and test cases are available. Traceability between software requirements and test cases is <u>completely missing</u>.</p> </div>
	<p>We claim that missing traceability indicates non-compliance of traceability with a standard.</p> <p>What is your opinion based on your practical experience? (Please select one)</p> <p style="text-align: center;">Strongly Agree Agree Disagree Strongly Disagree Don't know</p>
	<p>Can you briefly explain why you agree or disagree?</p>
	<p>We claim that missing traceability indicates safety risks.</p> <p>What is your opinion based on your practical experience? (Please select one)</p> <p style="text-align: center;">Strongly Agree Agree Disagree Strongly Disagree Don't know</p>
	<p>Can you briefly explain why you agree or disagree?</p>

Exemplary assessment report for the 2nd problem class:



```

Traceability Compliance Report
***** Completely Missing Traceability *****
Assessed Safety Standard: DO-178-B
Assessed Software Project: GeneAuto

Traceability required in DO-178-B:
>>> System Requirements Data <=> Software Requirements Data <<<
>>> System Requirements Data <=> Software Requirements Data <=> Low-level Requirements Data <<<
>>> Software Requirements Data <=> Software Architecture <<<
>>> Software Requirements Data <=> Low-level Requirements Data <<<
>>> Software Requirements Data <=> Low-level Requirements Data <=> Source Code <<<
>>> Software Requirements Data <=> Software Verification Case <<<
>>> Source Code <=> Low-level Requirements Data <<<
>>> Source Code <=> Low-level Requirements Data <=> Software Verification Case <<<

Completely missing traceability in GeneAuto:
>>> System Requirements Data <=>MISSING=> Software Requirements Data <<<
>>> System Requirements Data <=>MISSING=> Software Requirements Data <=>MISSING=> Low-level Requirements Data <<<
>>> Software Requirements Data <=>MISSING=> Software Architecture <<<
>>> Software Requirements Data <=> Low-level Requirements Data <=>MISSING=> Source Code <<<
>>> Software Requirements Data <=>MISSING=> Software Verification Case <<<
>>> Source Code <=>MISSING=> Low-level Requirements Data <<<
>>> Source Code <=>MISSING=> Low-level Requirements Data <=>MISSING=> Software Verification Case <<<

Assessment summary:
>>> Total number of required traceability: 8 <<<
>>> Number of completely *MISSING* traceability: 7 <<<
>>> Percentage of completely *MISSING* traceability: 88 % <<<
***** Completely Missing Traceability *****
    
```



Is the reported information helpful for a **certifier** to check the compliance of **GeneAuto's** traceability with the **DO-178B** standard?
(Please select one answer)

Strongly Agree Agree Disagree Strongly Disagree Don't know



Can you briefly explain why you agree or disagree?



Is the reported information helpful for a **project participant** to ensure the compliance of **GeneAuto's** traceability with the **DO-178B** standard?
(Please select one answer)


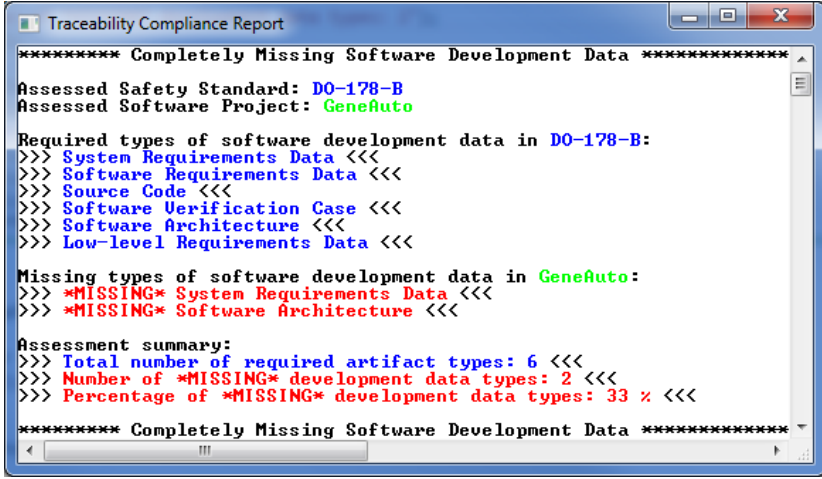






Strongly Agree Agree Disagree Strongly Disagree Don't know



Can you briefly explain why you agree or disagree?

Your Opinion - 3rd Problem Class

	<p>A type of software development data for which traceability is prescribed is completely missing.</p>
	<p>Illustrating example:</p> <div style="display: flex; justify-content: space-between;"> <div data-bbox="448 443 868 591"> <p>Standard</p> </div> <div data-bbox="879 443 1262 591"> <p>The DO-178B standard prescribes: “Software architecture, developed during the design process, should be <i>traceable</i> to software requirements”.</p> </div> </div> <hr/> <div style="display: flex; justify-content: space-between;"> <div data-bbox="448 600 868 958"> <p>Project</p> </div> <div data-bbox="879 600 1262 837"> <p>Situation within the project: Software requirements data are available, software architecture data are completely <i>missing</i>. This implies that prescribed traceability can not be created between software requirements data and the missing software architecture data.</p> </div> </div>
	<p>We claim that missing software development data, for which traceability is prescribed, indicate non-compliance of traceability with a standard. What is your opinion based on your practical experience? (Please select one)</p> <p style="text-align: center;">Strongly Agree Agree Disagree Strongly Disagree Don't know</p>
	<p>Can you briefly explain why you agree or disagree?</p>
	<p>We claim that missing software development data, for which traceability is prescribed, indicate safety risks. What is your opinion based on your practical experience? (Please select one)</p> <p style="text-align: center;">Strongly Agree Agree Disagree Strongly Disagree Don't know</p>
	<p>Can you briefly explain why you agree or disagree?</p>

	<p>Exemplary assessment report for the 3rd problem class:</p> 
	 <p>Is the reported information helpful for a certifier to check the compliance of GeneAuto's traceability with the DO-178B standard? (Please select one answer)</p> <p>Strongly Agree Agree Disagree Strongly Disagree Don't know</p>
	<p>Can you briefly explain why you agree or disagree?</p>
	 <p>Is the reported information helpful for a project participant to ensure the compliance of GeneAuto's traceability with the DO-178B standard? (Please select one answer)</p> <p>Strongly Agree Agree Disagree Strongly Disagree Don't know</p>
	<p>Can you briefly explain why you agree or disagree?</p>

Your Opinion - 4th Problem Class

	<p>Traceability is available but <i>deviates</i> from the prescription.</p>
	<p>Illustrating example:</p>
	<div style="display: flex; justify-content: space-between;"> <div data-bbox="443 414 869 604"> <p>Standard</p> <p>Traceability is prescribed with intermediate design descriptions source code → design description → software requirements</p> </div> <div data-bbox="877 414 1276 683"> <p>The DO-178B standard prescribes: “Traceability between the source code and software requirements throughout design descriptions should be provided to give visibility to the design decision made during the design process and to allow verification of the complete implementation of the software requirements.”</p> </div> </div>
	<div style="display: flex; justify-content: space-between;"> <div data-bbox="443 716 869 1019"> <p>Project</p> <p>Traceability is available but not through design descriptions</p> <p>All software requirements (R₁, R₂) are traceable to source code but not through design descriptions</p> <p>All software codes (SC₁, SC₂) are traceable to software requirements but not through design descriptions</p> </div> <div data-bbox="877 716 1276 1019"> <p>Situation within the project: Software requirements, design descriptions, and source code are available. Traceability between software requirements and source code is available. Traceability <i>deviates</i> from the standard because it does not include design descriptions. Thus, it cannot be used to give visibility to the design decision.</p> </div> </div>
	<p>We claim that deviating traceability indicates non-compliance with a standard. What is your opinion based on your practical experience? (Please select one)</p> <p>Strongly Agree Agree Disagree Strongly Disagree Don't know</p>
	<p>Can you briefly explain why you agree or disagree?</p>
	<p>We claim that deviating traceability indicates safety risks. What is your opinion based on your practical experience? (Please select one)</p> <p>Strongly Agree Agree Disagree Strongly Disagree Don't know</p>
	<p>Can you briefly explain why you agree or disagree?</p>



Exemplary assessment report for the 4th problem class:

```

Traceability Compliance Report
***** Deviating Traceability *****
Assessed Safety Standard: DO-178-B
Assessed Software Project: GeneAuto

Traceability required in DO-178-B:
>>> System Requirements Data <=> Software Requirements Data <<<
>>> System Requirements Data <=> Software Requirements Data <=> Low-level Requirements Data <<<
>>> Software Requirements Data <=> Software Architecture <<<
>>> Software Requirements Data <=> Low-level Requirements Data <<<
>>> Software Requirements Data <=> Low-level Requirements Data <=> Source Code <<<
>>> Software Requirements Data <=> Software Verification Case <<<
>>> Source Code <=> Low-level Requirements Data <<<
>>> Source Code <=> Low-level Requirements Data <=> Software Verification Case <<<


Deviating traceability in software development project GeneAuto:
>>> Software Requirements Data <=>DEVIATED=> Source Code
From: Software Requirements Data <=> Low-level Requirements Data <=> Source Code <<<

Assessment summary:
>>> Total number of required traceability: 8 <<<
>>> Number of deviating traceability: 1 <<<
>>> Percentage of deviating traceability: 12 % <<<

***** Deviating Traceability *****

```




 Is the reported information helpful for a **certifier** to check the compliance of **GeneAuto's** traceability with the **DO-178B** standard? (Please select one answer)

Strongly Agree Agree Disagree Strongly Disagree Don't know



Can you briefly explain why you agree or disagree?



 Is the reported information helpful for a **project participant** to ensure the compliance of **GeneAuto's** traceability with the **DO-178B** standard? (Please select one answer)

Strongly Agree Agree Disagree Strongly Disagree Don't know



Can you briefly explain why you agree or disagree?

Your Opinion - 5th Problem Class

	<p>Traceability is available but with <i>alternative routes</i>.</p>
	<p>Illustrating example:</p> <div style="display: flex; justify-content: space-between;"> <div style="width: 45%;"> <p>Standard</p> <p>Traceability is prescribed with intermediate design descriptions source code → design description → software requirements</p> </div> <div style="width: 45%;"> <p>The DO-178B standard prescribes: “Traceability between the source code and software requirements throughout design descriptions should be provided to give visibility to the design decision made during the design process and to allow verification of the complete implementation of the software requirements.”</p> </div> </div> <hr/> <div style="display: flex; justify-content: space-between;"> <div style="width: 45%;"> <p>Project</p> <p>Alternative routes for traceability: (1) source code → software requirement (2) source code → design description → software requirement</p> <p>R_1 is traceable to a source code through a different route than R_2</p> <p>SC_1 is traceable to a software requirement through a different route than SC_2</p> </div> <div style="width: 45%;"> <p>Situation within the project: Software requirements, design descriptions, and source code are available. Traceability between software requirements and source code is available. Though, the project provides <i>ambiguous</i> traceability: some source codes are directly traced to software requirements, other source codes are traced through design descriptions to software requirements.</p> </div> </div>
	<p>We claim that <i>ambiguous traceability</i> indicates non-compliance of traceability. What is your opinion based on your practical experience? (Please select one)</p> <p style="text-align: center;"> <input type="radio"/> Strongly Agree <input type="radio"/> Agree <input type="radio"/> Disagree <input type="radio"/> Strongly Disagree <input type="radio"/> Don't know </p>
	<p>Can you briefly explain why you agree or disagree?</p>
	<p>We claim that <i>ambiguous traceability</i> indicates safety risks. What is your opinion based on your practical experience? (Please select one)</p> <p style="text-align: center;"> <input type="radio"/> Strongly Agree <input type="radio"/> Agree <input type="radio"/> Disagree <input type="radio"/> Strongly Disagree <input type="radio"/> Don't know </p>
	<p>Can you briefly explain why you agree or disagree?</p>



Exemplary assessment report for the 5th problem class:

```

Traceability Compliance Report
***** Ambiguous Traceability *****
Assessed Safety Standard: DO-178-B
Assessed Software Project: GeneAuto

Traceability required in DO-178-B:
>>> System Requirements Data <=> Software Requirements Data <<<
>>> System Requirements Data <=> Software Requirements Data <=> Low-level Requirements Data <<<
>>> Software Requirements Data <=> Software Architecture <<<
>>> Software Requirements Data <=> Low-level Requirements Data <<<
>>> Software Requirements Data <=> Low-level Requirements Data <=> Source Code <<<
>>> Software Requirements Data <=> Software Verification Case <<<
>>> Source Code <=> Low-level Requirements Data <<<
>>> Source Code <=> Low-level Requirements Data <=> Software Verification Case <<<

Ambiguous traceability in software development project GeneAuto:
>>> Software Requirements Data <=>#AMBIGUOUS#> Source Code
1. route: Software Requirements Data <=> Source Code
2. route: Software Requirements Data <=> Low-level Requirements Data <=> Source Code <<<

Assessment summary:
>>> Total number of required traceability: 8 <<<
>>> Number of ambiguous traceability: 1 <<<
>>> Percentage of ambiguous traceability: 12 % <<<
***** Ambiguous Traceability *****
  
```



Is the reported information helpful for a **certifier** to check the compliance of **GeneAuto's** traceability with the **DO-178B** standard?
(Please select one answer)

Strongly Agree Agree Disagree Strongly Disagree Don't know



Can you briefly explain why you agree or disagree?



Is the reported information helpful for a **project participant** to ensure the compliance of **GeneAuto's** traceability with the **DO-178B** standard?
(Please select one answer)

Strongly Agree Agree Disagree Strongly Disagree Don't know



Can you briefly explain why you agree or disagree?

Selbstständigkeitserklärung

Ich versichere, dass ich die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus anderen Quellen direkt oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet.

Weitere Personen waren an der inhaltlich-materiellen Erstellung der vorliegenden Arbeit nicht beteiligt. Insbesondere habe ich hierfür nicht die entgeltliche Hilfe von Vermittlungs- bzw. Beratungsdiensten (Promotionsberater oder anderer Personen) in Anspruch genommen. Niemand hat von mir unmittelbar oder mittelbar geldwerte Leistungen für Arbeiten erhalten, die im Zusammenhang mit dem Inhalte der vorgelegten Dissertation stehen.

Die Arbeit wurde bisher weder im In- noch im Ausland in gleicher oder ähnlicher Form einer Prüfungsbehörde vorgelegt.

Ich bin darauf hingewiesen worden, dass die Unrichtigkeit der vorstehenden Erklärung als Täuschungsversuch angesehen wird und den erfolglosen Abbruch des Promotionsverfahrens zu Folge hat.

Ilmenau, 29.06.2015

Patrick Rempel