Published in Proceedings of the the 36th International Conference on Software Engineering (ICSE), Hyderabad, India

@INPROCEEDINGS{conf/icse/RempelMKC14, author = {Patrick Rempel and M{\"a}der, Patrick and Tobias Kuschke and Jane Cleland-Huang}, title = {Mind the Gap: Assessing the Conformance of Software Traceability to Relevant Guidelines}, booktitle = {Proc. of the 36th International Conference on Software Engineering {(ICSE)}, Hyderabad, India}, year = {2014}}

Mind the Gap: Assessing the Conformance of Software Traceability to Relevant Guidelines

Patrick Rempel[1], Patrick Mäder[1], Tobias Kuschke[1], and Jane Cleland-Huang[2]

Technische Universität Ilmenau[1]
Software Systems/Process Informatics Group
Ilmenau, Germany
{patrick.rempel|patrick.maeder}@tu-ilmenau.de

DePaul University[2]
Systems and Requirements Engineering Center
Chicago, IL, USA
jhuang@cs.depaul.edu

ABSTRACT

Many guidelines for safety-critical industries such as aeronautics, medical devices, and railway communications, specify that traceability must be used to demonstrate that a rigorous process has been followed and to provide evidence that the system is safe for use. In practice, there is a gap between what is prescribed by guidelines and what is implemented in practice, making it difficult for organizations and certifiers to fully evaluate the safety of the software system. In this paper we present an approach, which parses a guideline to extract a Traceability Model depicting software artifact types and their prescribed traces. It then analyzes the traceability data within a project to identify areas of traceability failure. Missing traceability paths, redundant and/or inconsistent data, and other problems are highlighted. We used our approach to evaluate the traceability of seven safety-critical software systems and found that none of the evaluated projects contained traceability that fully conformed to its relevant guidelines.

Categories and Subject Descriptors

D.2.9 [**Software Engineering**]: Management—*Life cycle*

General Terms

Documentation, Management

Keywords

Software traceability, safety, standard, guideline, compliance, conformance, certification, software and system safety, safety critical, failure patterns, assessment, inspection

1. INTRODUCTION

Developing software for regulated industries is a challenging process. Not only must the software deliver the required features, but it must do so in a way that ensures that the system is safe and secure for its intended use [27, 10]. To this end safety-critical, and other regulated systems, must meet stringent guidelines before they can be approved or certified for use [50, 19, 3]. For example, software developed for the aerospace industry must comply to the ISO12207 and/or the DO-178B guidelines, while software developed for European railway communication, signaling, and processing systems, must comply to EN50128 [13, 20]. Most guidelines prescribe a set of steps, deliverable documents, and exit criterion focused around planning, analysis and design, implementation, verification and validation, configuration management, and quality assurance activities. In addition they often provide specific guidelines for the creation and use of traceability in the project. For example, depending upon the criticality level of a requirement, the US Federal Aviation Authority guideline DO-178B requires traceability from requirements to design, and from requirements to source code and executable object code [50].

In practice, traceability is achieved through the creation and use of *trace links*, defined by the Center of Excellence for Software and Systems Traceability (CoEST) as "specified associations between a pair of artifacts, one comprising the source artifact and one comprising the target artifact" [26, 12, 11]. Software traceability serves an important role in demonstrating that a delivered software system satisfies its software design constraints and mitigates all identified hazards. When executed correctly, traceability demonstrates that a rigorous software development process has been established and systematically followed.

1.1 Traceability Challenges

Organizations struggle to establish and maintain accurate and complete sets of traceability links [5, 38, 49, 48, 39]. A prior 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, 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" [21], and the traceability data documented in the submissions [39]. While all of the submissions made some attempt 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 links, and problems in trace granularity which made it very difficult to understand the rationale for individual links. These observations draw attention to the *traceability gap* which exists between the traceability specified and required in guidelines and the actual trace links established in

ICSE '14, May 31 - June 7, 2014, Hyderabad, India

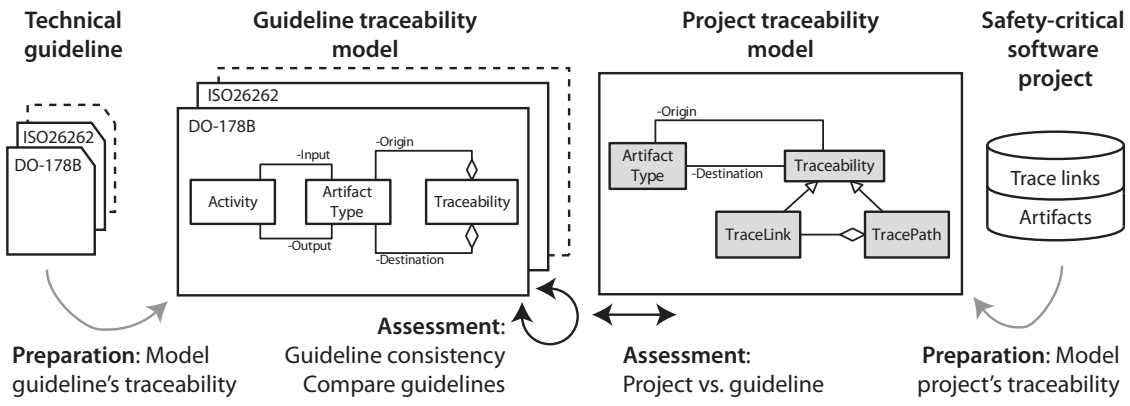

Figure 1: Overview of the proposed assessment approach

software projects. Trace deficiencies make it difficult for the manufacturers to ensure that the system they are building is safe for use and for certifiers to understand whether they should approve it for use in the public domain.

Furthermore, the emerging trend towards the adoption of agile methods in regulated domains introduces the need for more continual evaluation. In fact, the European Open-DO initiative [42], actively seeks to address the *Big Freeze* problem in which the significant cost and effort of the certification process makes it difficult to introduce change once the product is certified. The initiative is developing techniques for integrating agility into the safety-critical software development process. Although many people do not consider agile techniques suitable for use in safety-critical domains, the verification-driven life-cycle based on a rigorous test-first environment with continuous integration of changes has been used effectively in a number of safety-critical projects [36, 32]. The Open-DO initiative has a specific goal to develop techniques and tools that will allow all software to be constantly maintained in a 'ready-to-certify' state. The big-bang approach in which trace links are created after-the-fact for certification purposes, obviously does not support this notion of constant certifiability. As traceability is a critical element of the certification process, our approach provides techniques and tools for constantly evaluating, and reporting on, the trace coverage of a project.

1.2 Gap Analysis

In this paper we provide formalisms, metrics, and tool support for evaluating traceability coverage with respect to the guidelines that are relevant for a specific project. Within the context of this paper, we define a technical guideline as a document describing a common position of industry and certification bodies on how to develop safety-critical software within a certain domain. Guidelines generally help to coordinate technical viewpoints of regulators and safety experts in their licensing practices. They serve as a reference for constructing a safety case and for demonstrating the safety of software based systems. Furthermore, they support manufacturers in developing products for international markets. In contrast regulations are rules of order having the force of law prescribed by a superior or competent authority. While a regulation only lists obligations that have to be fulfilled in order to be compliant with law, a guideline suggests measures that should be taken in order to provide evidence that a software complies with regulations.

Performing a traceability gap analysis is similar to the notion of using test coverage tools to measure the extent to which unit test cases cover lines of code, decisions and branches, methods, and classes [57]. While test coverage tools are designed to evaluate the rigor of the testing process and to predict the readiness of the software for release, our trace coverage tool assesses the extent to which the created trace links adhere to the guidelines for the project. Later in this paper, we introduce a set of ten trace coverage metrics that measure traceability coverage at the guideline, the project, and the artifact level.

1.3 Our Approach

Our approach can be applied during the process of preparing the system for initial certification or continuously throughout the software development life-cycle in order to achieve continuous certification [42]. It involves three primary steps. First, those parts of a guideline that are relevant to the traceability of a complying development project are translated into a formal representation. This is a manual step that produces a formal model of a guideline, which can be reused across any impacted project. Second, project data is automatically parsed and relevant information, such as artifact identifiers, artifact types, and trace links, are captured in a formal representation. Finally, formally specified rules are used to analyze the data for traceability problems within an individual guideline, between guidelines, and between a relevant guideline and project data. The concepts of our approach as well as its major steps are illustrated in Figure 1. The guideline-level and project-level models serve as meta-models for representing traceability within individual guidelines and projects.

The remainder of the paper is laid out as follows. Section 2 describes five major usage scenarios of our work. Section 3 describes the proposed approach for modeling and assessing guidelines. Section 4 describes the preparation and assessment of project data as well as the problem types that can be identified in that data. Section 5 evaluates the different usage scenarios against industrial guidelines and relevant safety-critical software projects. In sections 6 and 7 we discuss the results and address threats to the validity of those results. Section 8 reviews related work in the area of requirements traceability and, more specifically, in measuring conformance between guidelines and project efforts. Finally, Section 9 concludes by proposing future research directions.

2. USAGE SCENARIOS

In this section, we discuss five major usage scenarios for our approach and outline their associated challenges.

Scenario 1: Project's Conformance to Guideline. For new projects, we provide traceability models that conform to the relevant guideline(s). These traceability models serves as a strategic plan for instrumenting the project environment and defining the processes needed to create, maintain, and use trace links as prescribed by the guideline. Furthermore, our process constantly evaluates the project for conformance to the traceability model.

Scenario 2: Continuous Certification. Consistently maintaining a project in a ready-to-certify state is challenging, but certainly not impossible [20]. It requires a rigorous verification process built into the development environment, continuous integration [18], and accurately maintained traceability, available at any time to support the certification process. Our approach supports ongoing analysis of the traceability coverage of a system with respect to the project's relevant guidelines.

Scenario 3: Integration of External Components. Safety-critical systems often integrate subsystems developed by different groups and subcontractors. Each of these subsystems carries its own traceability information. However, at integration time, traceability must be demonstrated across subsystems. The traceability model extracted from the guidelines helps to highlight the required integration-level trace coverage, and to evaluate its actual coverage in the integrated project. For example, it could take a high-level requirement which is realized across multiple subsystems, and measure and display its overall traceability coverage.

Scenario 4: 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 (e.g. DO-178B → DO-178C) and the new version becomes immediately 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 performed between the updated and original traceability model. Our approach provides support for identifying trace deficiencies introduced by the adoption of a new or revised guideline.

Scenario 5: Multiple Guideline Conformance. 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 [23]. Our systematic process provides a single set of traceability requirements, which can be used for planning purposes and for evaluating trace coverage.

These five scenarios provide the context for work described in the remainder of this paper.

3. GUIDELINE ANALYSIS

In this study, we analyzed five representative technical guidelines (DO-178B [50], ISO 26262-6 [31], ECSS-E-40 [19], FDA [21]), with application to four industrial domains: Aviation, Space, Automotive, and Medical. These guidelines were selected based on a literature review and are discussed in more detail in Section 5.1. The first step of the proposed approach involves manually creating a model of the traceability concepts prescribed by a guideline. All studied guidelines follow a similar structure which starts with a description of the development life-cycle. This life-cycle consists of multiple processes, each composed of activities, their prerequisites, and the artifacts they are required to produce. Additionally, a guideline defines objectives to be fulfilled for demonstrating safety. One of the most important objectives is software traceability. In this section, we use the DO-178B guideline to illustrate our approach. Guidelines were modeled manually by one author of this paper. The process involved reading the guidelines, identifying artifacts and their required traceability links, and then specifying them formally using F-Logic.

3.1 F-logic

For the formalization, representation, and assessment of the traceability concepts found in the guidelines and the projects' artifact and traceability structure we used the F-logic (frame logic) language. We chose F-logic because it is a knowledge representation and ontology language, which combines the advantages of conceptual modeling with object-oriented, frame-based languages and offers a declarative and compact syntax [34]. Features include object identity, complex objects, inheritance, polymorphism, query methods, and encapsulation. F-logic allowed us to define the required complex data schema, to reason about differences between two schemas, to reason about differences between a schema and instances, and to infer implicit knowledge from a schema.

As elaborated by Kifer et al. [35], the F-logic language consists of the sets $\mathcal{C}$ (*object constructors*) and $\mathcal{V}$ (*variables*), logical connectives and quantifiers ($\vee, \wedge, \neg, \leftarrow, \forall, \exists$), and auxiliary symbols ($[,], (,), \{, \}, \Rightarrow, \Rightarrow\!\!>, \rightarrow, \twoheadrightarrow, \bullet\!\!\rightarrow, \bullet\!\!\twoheadrightarrow, :$, ::, :=:, etc.).

Classes are defined as a group of objects or as a subclass of another class. For example:

```
Requirement   ::  Artifact.
        R23   :   Requirement.          (1)
        R23   :=: R25.
```

expresses that `Requirement` is a *subclass* of `Artifact`, that `R23` *is-a* or an *instance-of* `Requirement`, and that `R23` is *equivalent* to `R25`.

Signature atoms specify constraints on classes. For example:

```
TraceLink[source ⇒ Artifact].
Activity[output ⇒⇒ ArtifactType].          (2)
```

expresses that the method expressions `source` and `output`, when applied to objects that belong to class `TraceLink` and `Activity` respectively, must yield an object belonging to class `Artifact` or `ArtifactType`. The first signature expression is a *scalar signature* (⇒), the second signature expression is a *set-valued signature* (⇒⇒).

Data atoms apply value-returning methods to objects. For example:

```
TL1 : TraceLink[source → ART1].
ACT1 : Activity[output →→ {AT1, AT2}].          (3)
```

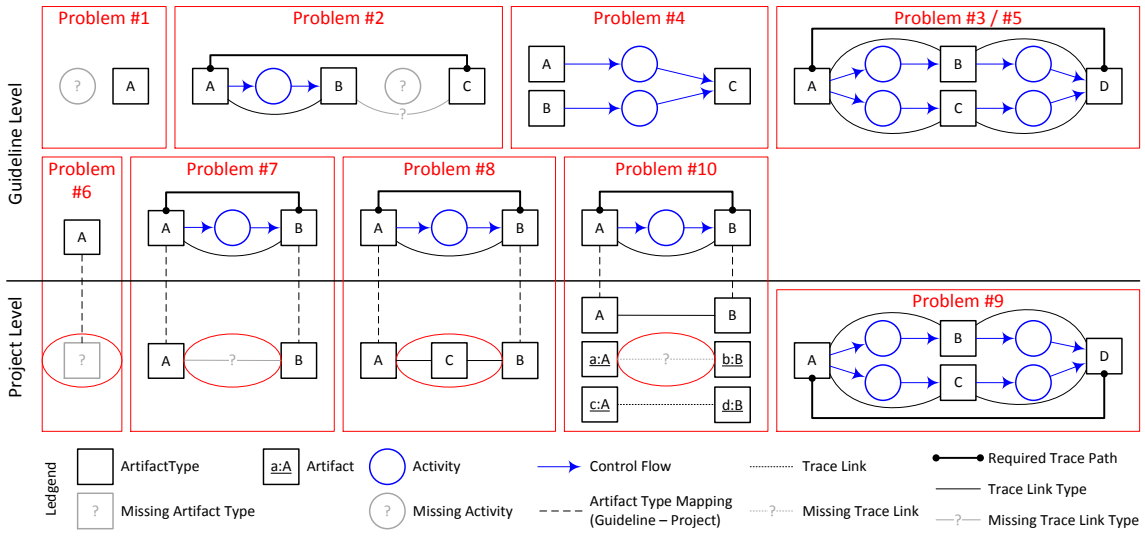

Figure 2: Problem types at guideline level (problems 1–5) and project level (problems 6–10)

expresses that the methods `source` and `output` are applied to the objects `TL1` and `ACT1` respectively. Method `source` returns the object `ART1` and method `output` returns the set of objects $\{\texttt{AT1}, \texttt{AT2}\}$. As with comparable logic-based languages, implication rules can be specified in the form of *head* $\leftarrow$ *body*, where *head* is a single *molecule* and *body* can be a conjunction of *molecules* or *negated molecules*. Thereby, a *molecule* is the simplest kind of formula constructed from the F-logic alphabet introduced above. The following example specifies an implication rule:

$$\texttt{?X} : \texttt{Activity}(\texttt{allArtifactTypes} \twoheadrightarrow \texttt{?Y}) \leftarrow \texttt{?X}(\texttt{input} \twoheadrightarrow \texttt{?I}, \texttt{output} \twoheadrightarrow \texttt{?O}) \wedge \texttt{?I}:\texttt{?Y} \wedge \texttt{?O}:\texttt{?Y}. \quad (4)$$

This rule states that the method `allArtifactTypes` returns for any object of class `Activity` a unified set of all `input` and `output` artifact types. The example also illustrates the syntax of variables (`?X`, `?Y`, `?I`, `?O`), named with a question mark and followed by letters, digits, or underscores.

3.2 Modeling the Guidelines

Formalizing a guideline involves modeling its required artifact types, activities, and traceability paths.

3.2.1 Step 1: Model Artifact Types

All investigated guidelines define types of artifacts, which are required to be created during software development. Guidelines also define which artifact type should include what information. Based on this information, types of artifacts that are required by a guideline can be derived (see Figure 1: `ArtifactType`).

Example: The DO-178B guideline [50] lists all required artifact types in a dedicated section. One required artifact type is software requirements data. The guideline states that *"Software Requirements Data is a definition of the high-level requirements including the derived requirements. This data should include functional and operational requirements under each mode of operation, performance criteria ..."*. The following molecule defines class `SWReqData` (Software Requirements Data) as a *subclass* of `ArtifactType`, which represents all artifact types.

$$\texttt{SWReqData} :: \texttt{ArtifactType}. \quad (5)$$

Additionally, the following *signature atom* asserts that class `SWReqData` consists of two multi-valued methods: `funcReq` (functional requirements) and `opReq` (operational requirements). Both methods correspond to the result type `SWReq` (software requirement) and are parameterized by `modeOfOp`, the related software requirement's mode of operation.

$$\texttt{SWReqData}[\texttt{funcReq@}(\texttt{modeOfOp}) \twoheadrightarrow \texttt{SWReq}, \texttt{opReq@}(\texttt{modeOfOp}) \twoheadrightarrow \texttt{SWReq}]. \quad (6)$$

Class `SWReq` consists of a single value method, which defines whether or not the software requirement is derived:

$$\texttt{SWReq}[\texttt{derived} \Rightarrow \texttt{boolean}]. \quad (7)$$

3.2.2 Step 2: Model Activities

Guidelines also define the activities of the prescribed software life-cycle (Figure 1: `Activity`). Each activity is characterized by the set of artifact types that it creates (Figure 1: $-$`Output`). Additionally, an activity may use one or more previously created artifact types (Figure 1: $-$`Input`).

Example: The DO-178B guideline [50] prescribes requirements specification as follows: "*Inputs to the software requirements process include the system requirements, the hardware interface and system architecture (if not included in the requirements) from the system life-cycle process, and the Software Development Plan, and the Software Requirements Standards from the software planning process. When the planned transition criteria have been satisfied, these inputs are used to develop the software high-level requirements.*" The following molecule asserts that class `SpecifySWReq` is a *subclass* of `Activity`.

$$\texttt{SpecifySWReq} :: \texttt{Activity}. \quad (8)$$

The following data molecule asserts that `SpecifySWReq` depends on the input artifact objects `SyRD` (system requirement data), `SyA` (system architecture), `SwDP` (software development plan), `SwRS` (software requirements standards) and produces the output `SWReqData`.

$$\texttt{SpecifySWReq}[\texttt{in} \twoheadrightarrow \{\texttt{SyRD}, \texttt{SyA}, \texttt{SwDP}, \texttt{SwRS}\}, \texttt{out} \twoheadrightarrow \{\texttt{SWReqData}\}]. \quad (9)$$

3.2.3 Step 3: Model Traceability Paths

A guideline also prescribes the traceability to be provided between artifact types (Figure 1: `Traceability`). Prescribed traceability is always related to an artifact type as origin (Figure 1: −`Origin`) and an artifact type as destination (Figure 1: −`Destination`).

Example: "*Traceability between source code and low-level requirements should be provided to enable verification of the absence of undocumented source code and verification of the complete implementation of the low-level requirements.*" The following data atom asserts that `RT1` is an element of the class `RequiredTrace` (required traceability), which requires traceability between `SWDesignSpec` (software design specification) and `SourceCode` (source code).

$$\texttt{RT1} : \texttt{RequiredTrace}[\texttt{origin} \rightarrow \texttt{SWDesignSpec}, \texttt{destination} \rightarrow \texttt{SourceCode}]. \quad (10)$$

3.3 Assessing Guideline Consistency

As illustrated in Figure 2 (problems #1, #2, and #3), we identified three types of problems that can occur in technical guideline models. We describe each problem and then discuss how it can be identified automatically from the formal model. For this purpose we provide formalized assessment rules per problem type.

Guideline Problem #1: Missing artifact creation activity. Every artifact type that is defined by a guideline should be created by a software process activity. Otherwise, artifacts of this artifact type cannot be traced to the activity that produced the artifact type nor to the artifact types that were used as input.

Implication: When conducting our case studies, we identified three potential reasons for the occurrence of this problem: (i) The definition of software process activities within the guideline is inconsistent; however we consider this unlikely given that technical guidelines are written and revised carefully by a group of subject matter experts. (ii) Activities or artifact types of the software life-cycle prescribed by a guideline have not been modeled correctly, leading to inconsistencies in the guideline model that are detected by our problem checking rules. (iii) A guideline is written within a larger context and refers to artifact types that were created in a process that is out of the guideline's scope. For instance, the guideline ISO 26262 [31] consists of ten parts, while only part six is related to software development. Furthermore, part six requires traceability to a system requirement artifact without defining system requirement creation activities.

Guideline Problem #2: Missing trace path. When a guideline requires traceability between two artifact types, it implicitly requires the existence of a direct or transitive trace path between both artifact types. This means that if no trace path exists between these two artifact types, it is clear at the definition level that the required traceability between the two artifacts cannot be satisfied.

Implication: As with to problem #1, this problem can be caused by an inconsistent guideline, an incorrect model creation, or an unresolved reference to a related guideline.

Guideline Problem #3: Ambiguous trace path. If a guideline requires traceability between two artifacts, the existence of a trace path between the artifact types is necessary (see problem #2). If multiple trace paths between two artifacts are identified, these paths are ambiguous.

Implication: Analogous to problems #1 and #2, the existence of this problem indicates an inconsistent guideline model caused by one of the three reasons described in the implications of problem #1. The main intention for creating the assessment rules for the problems #1 to #3 was to automatically evaluate the consistency of the derived guideline model and to thereby assure the quality of the model preparation as described in Section 3.

3.4 Assessing Merged Guidelines

We identified two problem types, which may be caused when two or more guidelines are relevant for a given product and must therefore be merged together. Figure 2 illustrates the two problems (#4 and #5) types.

Guideline Problem #4: Contradicting artifact creation after merge. Every single guideline defines software process activities and artifact types, which are produced by activities. However, if two guidelines define different activities for creating the same artifact type, these activities contradict each other.

Implication: The existence of this problem type indicates that two or more guidelines cannot be unified into a single model which respects all of the stated guidelines. If a software products needs to comply to both guidelines, measures must be taken to explicitly distinguish contradicting artifact type creation activities. Otherwise, the origin of a created artifact cannot be determined unambiguously.

Guideline Problem #5: Ambiguous trace path after merge. Every single guideline defines required traceability. Additionally, trace paths can be derived from the defined software process activities. If two guidelines require traceability between the same pair of artifact types but prescribe different trace paths, the merging of these guidelines introduces ambiguous trace paths to the merged model.

Implication: As with the previous problem type, the analyzed guidelines cannot be unified into one model respecting all requirements.

4. PROJECT COMPLIANCE

The primary purpose of our work is to analyze compliance of an individual project's traceability to its relevant set of guidelines. Therefore, we also need to formally model the traceability artifacts found in an individual project and to compare these to the formal model of the guidelines. We first describe our process for constructing the project level Traceability Model.

4.1 Constructing a Project Level Model

The constructed model consists of traceability artifacts, traceability paths, and traceability links.

4.1.1 Step 1: Parse Project Artifacts and Trace Links

The first step towards a project level traceability model is to collect all artifacts and trace links from within the software project. Typical representations of artifacts are text documents, hypertext, application database records, source code, and diagrams. Trace links typically appear in the form of trace link matrices, hyper links, text references to artifact identifiers, database records, and through the re-use of artifact names. Once, all project artifacts and trace links are

Table 1: Problem types and assessment rules to detect them

	#	Problem	Assessment rule
Guideline level	1	Missing artifact creation activity	Holds true iff neither the artifact type is defined as output of at least one activity nor the artifact type is a direct or transitive part of at least one artifact type that is defined as output of at least one activity. ▷ `missing_create_activity(?X)` ⟵ `?X : ArtifactType[isOut → 1]` ∧ `?Y : Activity` ∧ `?Y[out →?X]`.
	2	Missing trace path	Holds true iff traceability is required between two artifacts but neither a direct nor a transitive trace path between the two artifact types can be derived from the artifacts graph. ▷ `missing_guideline_path(?X, ?Y)` ⟵ `required_path(?X, ?Y)` ∧ `guideline_path(?X, ?Z)` ∧ ¬ (`?Z :?Y`).
	3	Ambiguous trace path	Holds true iff traceability is required between two artifact types but more then one direct or transitive trace paths between the two artifact types can be derived from the artifacts graph. ▷ `ambiguous_guideline_path(?X, ?Y)` ⟵ `required_path(?X, ?Y)` ∧ `?I = count{?Z|Z : guideline_path(?X, ?Z)}` ∧ `?I > 1`.
	4	Contradicting artifact creation after merge	Holds true iff two different activities create the same artifact type as output after merging two guidelines. ▷ `contradicting_activity(?X, ?Y, ?Z)` ⟵`?X[out →?Z]`∧`?Y[out →?Z]` ∧ ¬ (`?X :=: ?Y`).
	5	Ambiguous trace path after merge	Holds true iff traceability is required between two artifacts but more then one direct or transitive trace paths between the two artifact types can be derived from the artifacts graph after merging two guidelines. ▷ *refer to problem #3*
Project level	6	Missing artifact type	Holds true iff an artifact type is required by the guideline but missing in the project. ▷ `missing_artifact_type(?X)` ⟵ `required_artifact_type(?X)`∧`project_artifact_type(?Y)`∧¬ (`?Y :?X`).
	7	Missing trace path	Holds true iff traceability is required between two artifacts by the guideline but neither a direct nor a transitive trace path between the two artifact types can be found in the project. ▷ `missing_project_path(?X, ?Y)` ⟵ `required_path(?X, ?Y)` ∧ `project_path(?X, ?Z)` ∧ ¬ (`?Z :?Y`).
	8	Invalid trace path	Holds true iff traceability is required between two artifacts by the guideline but the trace path on guideline level varies from the trace path on project level. ▷ `invalid_path(?X, ?Y)` ⟵ `?Z : project_path(?X, ?Y)[hash →?P]`∧`?Q : guideline_path(?X, ?Y)[hash →?R]`∧`?P != ?R`.
	9	Ambiguous trace path	Holds true iff traceability is required between two artifact types by the guideline but more then one direct or transitive trace paths between the two artifact types can be derived from the project. ▷ `ambiguous_project_path(?X, ?Y)` ⟵ `required_path(?X, ?Y)` ∧ `?I = count{?Z|Z : project_path(?X, ?Y)}` ∧ `?I > 1`.
	10	Missing trace link	Holds true iff traceability between two artifact types is required by the guideline but the project misses a trace link between two artifacts of the required artifact types. ▷ `missing_link(?X, ?Y)` ⟵ `required_link(?X, ?Y)` ∧ `project_link(?X, ?Z)` ∧ ¬ (`?Z :?Y`).

identified, the raw artifact and trace data needs to be parsed and transformed into our standardized format for further processing steps. As described in Section 5.3 we developed a number of automated parsers to perform this task.

4.1.2 Step 2: Extract Artifacts and Trace Types

Once all data related to artifacts and trace links has been preprocessed, artifact and trace link types can be assigned to the data records. By aggregating artifacts and trace links by type, a project traceability model can be derived.

4.1.3 Step 3: Map Guideline and Project Model

The project traceability model, created in the previous step, contains information about a project's artifact types, trace link types, and trace paths at the same abstraction level as the guideline traceability model (see Section 3). To assess whether or not a project fulfills the traceability requirements defined by a guideline, artifact types of the project traceability model need to be mapped to the corresponding artifact types of the guideline traceability model.

4.2 Assessing Project Data vs. Guideline

By comparing the modeled project data against a guideline model the following problem types can be identified.

Compliance Problem #6: Missing artifact type. The artifact types required by a guideline must be available within a project in order for traceability to be established between them. To check this necessary precondition, the completeness of artifact types at the project level is assessed.

Compliance Problem #7: Missing trace path. In additional to problem #6, traceability required by a guideline between artifact types can only be established at the project level if a trace path between the two artifact types is available at the project level.

Compliance Problem #8: Invalid trace path. A trace path required by a guideline must not merely exist at the project level, but its route must also conform to the prescription of the guideline. To assess this problem, the artifact route of a trace path at project level can be compared with the artifact route of a trace path at the guideline level.

Compliance Problem #9: Ambiguous trace path. Traceability is ambiguous if multiple trace paths between two artifact types can be identified (compare problem #3).

Compliance Problem #10: Missing trace link. In addition to problem #7, traceability between artifact types that is required by a guideline can only be considered complete if at the project level every single artifact of the requested source artifact type is traced directly via a trace link or transitively via a trace path to an artifact of the requested target artifact type. We consider every required but non-existing direct or transitive trace link a missing trace.

5. CASE STUDIES

In order to evaluate our approach we conducted several case studies that refer back to the usage scenarios described

in Section 2. We consider the support of Scenario 1 (Support for Project Planning) as a by-product of our approach. Once, the traceability requirements of a guideline have been successfully modeled (see Section 3.3) our prototype is able to generate a traceability model that conforms to selected guidelines. This model can be used by stakeholders as they establish a traceability plan for their project. Supporting Scenario 2 (Constant Certifiability) and Scenario 3 (Integration of External Components) require a similar assessment procedure, which compares a project's artifacts and trace links against applicable guidelines. In order to demonstrate the capabilities of our approach in supporting those usage scenarios, we assessed seven safety-critical software projects against their respective technical guidelines. Supporting Scenario 4 (Migration to New or Revised Guideline) and Scenario 5 (Multiple Guideline Conformance) also require a similar assessment procedure, which compares two guideline models with each other. We conducted a second analysis in which we performed five comparisons among the five technical guidelines relevant for the assessed projects in the previous case study. The five guidelines and seven safety-critical projects for those studies are briefly introduced in the following subsections.

5.1 Technical Guidelines

Throughout the case studies, we use five technical guidelines applicable for the development of safety-critical software in different domains. Table 2 shows these guidelines and each of their application domains. It also provides a quantitative overview in terms of relevant concepts for our approach. The number of required artifact types provides an impression of the complexity of the described development process per guideline, while the number of trace paths refers to requested traceable connections between a pair of artifact types, which may be connected through a single trace link or a chain of trace links. All five guidelines were modeled according to the process described in Section 3. Each model was assessed for problems #1 to #3 to ensure its correctness and consistency. Once all problems were resolved, all models were reviewed by a colleague with extensive experience in safety-critical development (not an author of this paper) for their accordance to the modeled guideline.

5.2 Assessed Projects

We analyzed seven different projects. Along with a short project description, we list relevant guidelines, the number of documented artifacts that were analyzed, and the number of analyzed, relevant trace links.

TOPCASED-SAM (TC-SAM) [56]. A subproject of the TOPCASED initiative, which provides a set of modeling, transformation and verifying tools for functional structured analysis. *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 (TC-REQ)[55]. Is a subproject of the TOPCASED initiative. The project's aim is developing a generic, tool independent way for ensuring traceability between requirements and model elements and for managing requirements. *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.

Table 2: Characteristics of assessed guidelines

Guideline	Domain	Relevant concepts
DO-178B/ ED-12B [50]	Aviation	31 artifact types 8 required trace paths
ISO 26262-6 [31]	Automotive	21 artifact types 4 required trace paths
ECSS-E-40 [19]	Space	73 artifact types 8 required trace paths
FDA Guide [21]	Medical	28 artifact types 10 required trace paths
TOPCASED quality kit (TC-QK)[53]	Cross domain	21 artifact types 1 required trace path

GeneAuto [22]. An open-source toolset for converting Simulink, Stateflow, and Scicos models into executable program code. 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.

Secure Auditing for Linux (SAL) [51]. Develops a kernel level auditing package for Red Hat Linux that is compliant with the Common Criteria specifications (C2 level equivalency) and provides features to protect logged information from unauthorized modification through encryption techniques. *Relevant guideline*: project dependent; *Artifacts:* 71 requirements, monolithic design description; *Relevant trace links:* 0 (other trace links exist).

Rate Adjustment by Managing Inflows (RAMI) [47]. Develops a TCP/IP flow control module for the Linux kernel and a suite of network evaluation utilities. *Relevant guideline*: project dependent; *Artifacts:* 48 requirements, 120 design description artifacts, source code functions 275, test cases not accessible; *Relevant trace links:* 553.

CONNECT [14]. An open-source software system and community that promotes IT interoperability in the U.S. healthcare system. CONNECT enables secure, electronic health data exchange among healthcare providers, insurers, government agencies, and consumer services. *Relevant guideline*: FDA Guide; *Artifacts:* 9 risks, 27 system requirements, 55 functional requirements, 27 non-functional requirements, 179 user stories, 6 technical stories, 10 epics, 290 improvements, 245 feature request, 775 bugs, 7 design artifacts, 1919 source code classes, 770 development tasks, 247 QA tasks, 70 research tasks, 254 infrastructure tasks, 170 documentation tasks; *Relevant trace links:* 16764.

Health Care Protocol Translator (HCPT) [28]. Provides a middleware system, which creates a bridge between different protocols, such as HL7 and DICOM. The system provides a web based interface, which allows users to search and synchronize information within one HIS or PACS system to another HIS or PACS system. *Relevant guideline*: FDA Guide; *Artifacts:* 12 use cases, 37 requirements, 4 model components, 41 model classes, 35 source code classes, 8 test cases, 14 test results; *Relevant trace links:* 189.

5.3 Prototype

We created a prototypical implementation that leverages the F-logic open source development environment Flora-2 [54]. We created traceability models for the five technical guidelines as well as inference rules to automatically as-

sess the captured data for the previously described problem types. Additionally, we developed crawlers to parse relevant project data (artifacts and trace links) from the seven studied software projects. Our prototype derives the project traceability model from the parsed data and transforms all information into Flora-2 expressions. By applying the developed reasoning rules per problem type (see Table 1) to the captured project and guideline related facts we were able to assess both the guidelines and compliance of each project against its relevant guidelines. Results from this evaluation are presented in the following subsections.

5.4 Project Assessment Results

Table 3 shows aggregated results of the projects versus guideline assessment. The first and second column refer to the project and the guideline that were compared. Columns three to seven refer to the five problem types that are relevant for this assessment. The first number within a cell shows the percentage of a problem's occurrence in relation to the total occurrence of the assessed concept within the project, while the number in brackets shows the problem occurrence count. Cells with a dash refer to problems that could not be assessed because preconditions in terms of artifacts or trace links were not available for the particular project. In Figure 3, we visualize the assessment results of project TC-SAM vs. guideline DO-178B.

Table 3: Assessment results per problem type for each project against its relevant guidelines

Assessment		Occurrence of problem type				
Project ⟷ Guideline		#6	#7	#8	#9	#10
TC-SAM	DO-178B	66% (4)	88% (7)	0%	0%	10% (8)
	ISO 26262-6	60% (3)	100% (4)	0%	0%	18% (45)
	ECSS-E-40	13% (1)	63% (5)	0%	0%	10% (8)
TC-REQ	DO-178B	66% (4)	88% (1)	0%	0%	0% (0)
	ISO 26262-6	60% (3)	100% (4)	0%	0%	0% (0)
	ECSS-E-40	13% (1)	63% (5)	0%	0%	0% (0)
GeneAuto	DO-178B	50% (3)	88% (7)	0%	0%	80% (167)
SAL	ECSS-E-40	38% (3)	100% (8)	0%	0%	0% (0)
RAMI	DO-178B	50% (3)	88% (7)	0%	0%	13% (74)
CONNECT	FDA Guide	0% (0)	70% (7)	0%	0%	9% (1657)
HCPT	FDA Guide	81% (9)	100% (10)	0%	0%	33% (62)

5.5 Guideline Assessment Results

Table 4 shows aggregated results of the guideline against guideline assessment. The first and the second column refer to the two guidelines A and B that were compared. Columns three to five refer to differences in terms of required artifact types between A and B. Column three shows the number of artifact types that are required in A, but not in B; column four refers to the number of artifact types that are required in A and B; and column five refers to the number of artifact types that are required in B, but not in A. Similarly, columns six to eight refer to differences in terms of required trace paths between both guidelines. We also assessed all five guideline combinations for problems #4 and #5 and found that neither problem types were present for the assessed guideline combinations.

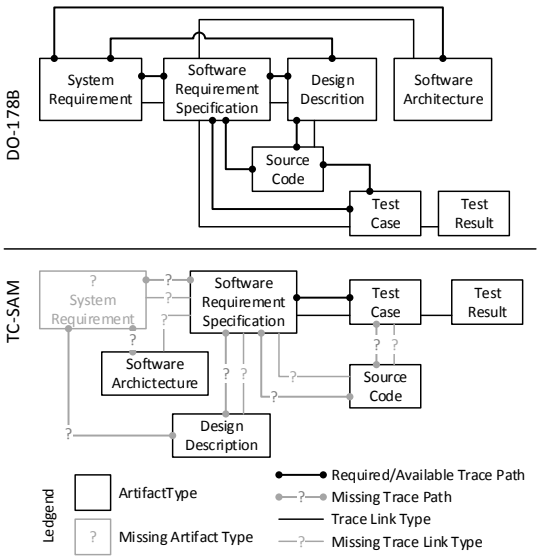

Figure 3: Guideline traceability model of DO-178B vs. project traceability model of TC-SAM

6. DISCUSSION

All studied cases were safety-critical software projects that are required to conform with one of the technical guidelines. The fact that our project assessment results (see Table 4) show that none of the projects provided sufficient traceability to conform with its relevant guideline(s) means that none of them can be considered as ready for certification.

6.1 Project Assessment

Before we start to discuss the results of every problem in detail, it should be noted that several problem types are interdependent. The existence of problem #6 means that not a single artifact of the missing artifact type exists at project level. This implies that not a single required trace link (problem #10) was established from or to an artifact of that artifact type. The correct amount of missing trace links can be calculated if the second artifact of the required trace link exists within the project. However, if source and target artifacts are both missing (problem #7), the actual number of missing trace links cannot be determined, because all required elements (source artifact, target artifact, trace) of the trace link are missing completely. Due to this dependency, problem #7 can be used as a leading indicator to detect critical areas of the project with respect to traceability. These problem interdependencies should be considered when interpreting the assessment results of a single project.

Missing artifacts (problem #10) indicate that trace links are incomplete at the project level and thus traceability analysis might lead to wrong assumptions and conclusions. To solve that issue, projects suffering from this problem need to add missing trace links for achieving trace link completeness at the project level with respect to the relevant guideline. For example, the GeneAuto project with more than 50% missing trace links, will require substantial effort to resolve this problem. RAMI and CONNECT also suffer from a high amount of missing trace links.

The absence of required trace paths at the project level (problem #7) indicates that required trace links between

Table 4: Comparison of guideline migrations in terms of required artifact types and trace paths

Guideline Comparison A ⟷ B		Artifact types (AT)			Trace paths (TP)		
		$A_{AT} \setminus B_{AT}$	$A_{AT} \cap B_{AT}$	$B_{AT} \setminus A_{AT}$	$A_{TP} \setminus B_{TP}$	$A_{TP} \cap B_{TP}$	$B_{TP} \setminus A_{TP}$
TC-QK	ISO 26262-6	0	3	3	0	1	3
ISO 26262	DO-178B	0	6	2	0	4	4
DO-178B	ECSS-E-40	0	8	1	0	8	1
DO-178B	FDA Guide	0	8	3	0	8	2
ECSS-E-40	FDA Guide	1	8	3	1	8	2

project artifacts of two types are missing. This means that required trace links must be established to conform with the guideline. As outlined in Table 4, no project has been setup with the capability of providing all the required trace paths. Most projects suffer from a high percentage (more than 70%) of missing trace paths at the project level.

The absence of required artifact types at the project level (problem #6) indicates that even the preconditions for establishing traceability conforming with a guideline are not fulfilled. Thus, artifacts of missing artifact types need to be created to enable a project to conform with a guideline.

Divergence of trace paths at the guideline and project level (problem #8) would indicate that even though traceability required by a guideline is available at the project level, it carries the wrong semantics and might not be usable to address the traceability goals of the guideline assigned to a specific trace path. The existence of ambiguous trace paths (problem #9) indicates that project artifacts cannot unambiguously be traced to their origin. As with problem #8, ambiguous trace paths at the project level carry wrong or at least ambiguous semantics and might not be usable to address related traceability goals. As a high percentage of the required trace paths were missing or incomplete for the evaluated projects, it is not surprising that problems #8 and #9 were not identified in the examined projects.

6.2 Guideline Assessment

The guideline assessment results shown in Table 3 demonstrate that differences among technical guidelines are relatively small in terms of required traceability. It becomes apparent that the TC-QK guideline, an in-house guideline for projects of the TOPCASED initiative, is a subset of ISO 26262-6, which is a subset of DO-178B. Similarly, DO-178B is a subset of the FDA Guide as well as ECSS-E-40. This implies that for a project, which complies to a complex guideline like the FDA Guide, no additional effort is required to conform with ISO 26262-6 or DO-178B. The contrary is true, if a project complies to a less restrictive guideline such as ISO 26262-6 and then needs to conform with ECSS-E-40, additional artifact types and trace paths will need to be added to the project.

7. THREATS TO VALIDITY

When planning and conducting our case studies we carefully considered validity concerns. The fact that our cases diverge across multiple guidelines and industrial projects of various sizes and domains suggests that our approach is applicable across a wide variety of projects. However, we are aware of the fact that a larger evaluation is required in order to gain generalizable results.

A potential threat exists due to the required studying and understanding of a guideline and the creation of the respective model. The responsible person may misunderstand or entirely miss requirements of a guideline. In order to mitigate that threat, we took three measures. First, we followed the same procedure for all analyzed guidelines, searching for the concepts: artifact type, activity, and required trace paths in subsequent passes. Second, we identified potential consistency problems that can occur within a guideline model and support their identification within our prototype. Third, all guideline models used for our case study were reviewed by a colleague with extensive experience in the development of safety-critical software.

Another potential threat exists in the preparation of the analyzed project data. We examined carefully the structure of each project and its available artifacts. However, the artifacts that we found were diverse and often spread across multiple tools and repositories. In order to identify possible problems, we performed where possible completeness and consistency checks on the captured data. Nonetheless, there remains a risk that we may have missed or misclassified certain artifacts or trace links. For ongoing work, we plan to involve project stakeholders in order to get feedback on the correctness and completeness of the captured data.

8. RELATED WORK

Several authors have conducted empirical research on requirements traceability and argued the need for planned traceability and defined traceability strategies. Gotel and Finkelstein showed that understanding the stakeholders responsible for the creation and maintenance of trace links is essential for improving the outcome of the tracing process [25]. Ramesh [45] identified two groups of traceability users, which he referred to as low-end and high-end users. While low-ends users rely on simple dependencies among requirements, high-end users leverage more sophisticated traceability schemes. Ramesh and Jarke [46] conducted a large practitioner and tool study on traceability. They pointed out that traceability links should be strongly typed in order to avoid semantic misinterpretations. As a result, the authors proposed a traceability meta-model and reference models as guidance for practitioners. Gotel et al. described the traceability life-cycle and highlighted the importance of planning and managing traceability in addition to the daily activities of creating, maintaining, and using trace links [24]. In prior work we also advocated the use of a traceability information model as a necessary condition to employ traceability [37, 39]. Arkley and Riddle [2] conducted a case study on a software project, which successfully leveraged traceability. They concluded that the success of the observed traceability system was mainly influenced by two facts: (i) general trace-

ability needs were examined to support project participants in their tasks, and (ii) the traceability information model was systematically tailored to the identified needs.

Several researchers investigated aspects of traceability in regulated industries. Mc Caffery et al. comprehensively discussed traceability requirements for medical device software development [9]. Hill and Tilley developed a database schema for tracing between a safety process improvement model, safety requirements and taxonomies, and software safety risks [29]. Peraldi-Frati and Albinet presented an approach for tracing non-functional requirements such as safety, timing, and performance across the life-cycle artifacts of an automotive system design [44]. Katta and Stalhane developed a very detailed traceability model depicting numerous artifact types and related links for safety-critical systems such as nuclear power stations [33]. Similarly, Sanchez et al. described a traceability meta-model and supporting techniques for integrating tracing into the model driven development of safety critical systems [52]. Borg et al. [4] investigated the reusability of traceability in safety critical control systems that had evolved over more than 30 years. However, the focus of all these publications was on prescribing general traceability techniques and practices as opposed to evaluating traceability compliance. One exception is our prior work [39], in which we reported on traceability problems that were observed by members of the U.S. Food and Drug Administration (FDA) responsible for evaluating traceability documentation as part of medical device approval. Several of the metrics described in this paper emerged from that study.

This paper presents a technique for measuring conformance of a project's traceability to relevant technical guidelines. To the best of our knowledge there have been no other efforts to create traceability metrics for evaluating actual traceability coverage. The closest work is that by Dinesh et al., who used natural language processing (NLP) techniques to extract formal process representations from regulatory documents. They used these process representations to analyze an organization's conformance to the regulation [17]. However, the authors focused on analyzing activities explicitly logged in a database. In contrast, our approach retrieves and analyzes artifacts and their associated trace links from project repositories, processes these to generate a structural model of traceability in the project and then compares this to the model prescribed in the guideline.

Researchers proposed solutions to formalize safety standards, regulations, and law for supporting compliance verification against those texts. De la Vara et al. [16] developed a generic meta-model for safety-standards, which is larger and more generic than our guideline model as its scope goes far beyond software traceability. Panesar-Walawege et al. [43] proposed a UML profile for ensuring software design compliance to the IEC61508 standard. Similarly, Nejati et al. defined a traceability information model, which links safety related requirements with software design components [41]. Maxwell and Antón [40] formalized legal texts of the Health Insurance Portability and Accountability Act as production rule models to support regulatory compliance analysis. Breaux et al. performed extensive research on formalizing legal text for supporting compliance verification of software requirements with regulations in order to establish traceability between requirements and regulations [6, 8, 7].

Furthermore, metrics have been proposed for a variety of other software engineering activities. For example, Ambriola and Gervasi developed a set of process metrics for evaluating stability and efficiency of the requirements analysis process [1]. Their metrics were designed to identify risky or inefficient behavior during the requirements analysis process, so that managers could take corrective actions. Other researchers and practitioners proposed and used test-coverage metrics for measuring and evaluating the quality of the testing process. These metrics are now used ubiquitously across many software development environments.

Finally, researchers developed techniques for reverse engineering software development processes through mining project repositories. For example, Hindle extracted software artifacts from software repositories and used these to infer the underlying software development process [30]. Similarly, Cook and Wolfe proposed a technique to instrument existing software systems to discover the software development process [15]. We also mine software repositories to extract traceability data. However, we use this data to infer specific problems with the traceability process rather than attempting to completely reconstruct the development process.

9. CONCLUSIONS AND FUTURE WORK

In this paper, we have focused primarily on the gap between the traceability that is prescribed by technical guidelines and the traceability that is implemented in practice. This gap makes it difficult for organizations and certifiers to fully evaluate the safety of developed software systems. We presented an approach, which uses a manually created model of a guideline's required traceability and related concepts to analyze project data and to identify areas of traceability failure. Furthermore, we defined ten types of traceability problems that may occur when a project aims to conform to one or more guidelines. Those problem types cover, for example, missing traceability paths as well as redundant and/or inconsistent data. Our approach facilitates the identification of those problems both during initial certification efforts and then continuously throughout the project's life cycle. As a byproduct we provide traceability models that conform to a guideline and can be used for planning a project's traceability. Furthermore, we support the comparison of guidelines in order to assist the migration of a project from compliance to an initial set of guidelines towards compliance to a revised or expanded set of guidelines.

We used our approach to evaluate the traceability of seven safety-critical software systems against a set of five technical guidelines and found that none of the evaluated projects was in full conformance to its relevant traceability guidelines. This suggests that these projects were not ready for certification and emphasizes the importance of performing a traceability analyses.

Future work will focus on the presentation of problems and on generalizing the access to project data for easier and continuous assessment. We will also extend our catalog of traceability problem classes and detection metrics. Finally, in addition to conformance of traceability data with technical guidelines, we are interested in extensions that unify various qualities into a single model.

10. ACKNOWLEDGMENTS

We are funded by the German Ministry of Education and Research (BMBF): grant 16V0116 and US National Science Foundation Grant CCF-1319680.

11. REFERENCES

[1] V. Ambriola and V. Gervasi. Process metrics for requirements analysis. In *Proc. of the 7th European Workshop on Software Process Technology (EWSPT), Kaprun, Austria*, pages 90–95, 2000.

[2] P. Arkley and S. Riddle. Tailoring traceability information to business needs. In *Proc. of the 14th IEEE International Requirements Engineering Conference (RE), Minneapolis/St. Paul, Minnesota, USA*, pages 239–244, 2006.

[3] BEL-V, BfS, CSN, ISTec, ONR, SSM, STUK. Licensing of safety critical software for nuclear reactors – common position of seven european nuclear regulators and authorised technical support organisations, 2013.

[4] M. Borg, O. C. Gotel, and K. Wnuk. Enabling traceability reuse for impact analyses: A feasibility study in a safety context. In *Proc. of the 7th International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE), San Francisco, USA*, pages 72–78. IEEE, 2013.

[5] E. Bouillon, P. Mäder, and I. Philippow. A survey on usage scenarios for requirements traceability in practice. In J. Doerr and A. L. Opdahl, editors, *Requirements Engineering: Foundation for Software Quality*, volume 7830 of *Lecture Notes in Computer Science*, pages 158–173. Springer, 2013.

[6] T. D. Breaux, A. I. Antón, and E. H. Spafford. A distributed requirements management framework for legal compliance and accountability. *Computers & Security*, 28(1):8–17, 2009.

[7] T. D. Breaux and D. G. Gordon. Regulatory requirements traceability and analysis using semi-formal specifications. In *Proc. of the 19th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ'13), Essen, Germany*, pages 141–157. Springer, 2013.

[8] T. D. Breaux and A. Rao. Formal analysis of privacy requirements specifications for multi-tier applications. In *Proc. of the 21st IEEE International Requirements Engineering Conference (RE), Rio de Janeiro, Brasil*, pages 14–23. IEEE, 2013.

[9] F. M. Caffery, V. Casey, M. Sivakumar, G. Coleman, P. Donnelly, and J. Burton. *Software and Systems Traceability*, chapter Medical Device Software Traceability, pages 321–339. Springer, 2011.

[10] CCMB-2006-09-001: Common criteria for information technology security evaluation: Part 1: Introduction and general model, v3.1 r1, 2006.

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

[12] CoEST: Center of excellence for software traceability, http://www.CoEST.org.

[13] C. Comar, F. Gasperoni, and J. Ruiz. Open-do: An open-source initiative for the development of safety-critical software. In *Proc. of the 4th IET International Conference on Systems Safety, London, UK*, pages 1–5. IET, 2009.

[14] CONNECT, developer.connectopensource.org, 2013.

[15] J. E. Cook and A. L. Wolf. Software process validation: quantitatively measuring the correspondence of a process to a model. *Transactions on Software Engineering and Methodology (TOSEM)*, 8(2):147–176, 1999.

[16] J. L. de la Vara and R. K. Panesar-Walawege. Safetymet: A metamodel for safety standards. In *Proc. of the 16th International Conference on Model Driven Engineering Languages and Systems (MODELS), Miami, USA*, pages 69–86. Springer, 2013.

[17] N. Dinesh, A. Joshi, I. Lee, and O. Sokolsky. Checking traces for regulatory conformance. In *Proc. of the 8th International Workshop on Runtime Verification (RV), Budapest, Hungary*, pages 86–103. Springer, 2008.

[18] P. Duvall, S. Matyas, and A. Glover. *Continuous Integration: Improving Software Quality and Reducing Risk*. Addison-Wesley, 2007.

[19] ECSS. ECSS-E-40C: principles and requirements applicable to space software engineering, 2009.

[20] P. Farail, P. Goutillet, A. Canals, C. Le Camus, D. Sciamma, P. Michel, X. Crégut, and M. Pantel. The TOPCASED project: a toolkit in open source for critical aeronautic systems design. *Ingenieurs de l'Automobile*, 1(781):54–59, 2006.

[21] Food and Drug Administration. *General Principles of Software Validation; Final Guidance for Industry and FDA Staff*, 2002.

[22] Gene-Auto, gforge.enseeiht.fr/projects/geneauto, 2013.

[23] D. G. Gordon and T. D. Breaux. A cross-domain empirical study and legal evaluation of the requirements water marking method. *Requirements Engineering*, 18(2):147–173, 2013.

[24] O. Gotel, J. Cleland-Huang, J. Hayes, A. Zisman, A. Egyed, P. Grünbacher, A. Dekhtyar, G. Antoniol, J. Maletic, and P. Mäder. Traceability fundamentals. In J. Cleland-Huang, O. Gotel, and A. Zisman, editors, *Software and Systems Traceability*, pages 3–22. Springer London, 2012.

[25] O. Gotel and A. Finkelstein. Extended requirements traceability: results of an industrial case study. In *Proc. of the 3rd IEEE Int. Symp. on Requirements Engineering (RE), Annapolis, USA*, 1997.

[26] O. Gotel and C. Finkelstein. An analysis of the requirements traceability problem. In *Proc. of the 1st IEEE Int. Conf. on Requirements Engineering (RE), Colorado Springs, USA*, pages 94 –101, apr 1994.

[27] W. S. Greenwell, E. A. Strunk, and J. C. Knight. Failure analysis and the safety-case lifecycle. In *Proc. of the 7th Working Conference on Human Error, Safety and Systems Development, Toulouse, France*, pages 163–176, 2004.

[28] Health Care Protocol Translator (HCPT), svn.assembla.com/svn/HITTeam, 2013.

[29] J. Hill and S. Tilley. Creating safety requirements traceability for assuring and recertifying legacy safety-critical systems. In *Proc. of the 18th IEEE Int. Requirements Engineering Conference (RE), Sydney, Australia*, pages 297–302, 2010.

[30] A. Hindle. Software process recovery: Recovering process from artifacts. In G. Antoniol, M. Pinzger, and E. J. Chikofsky, editors, *Proc. of the 17th Working Conference on Reverse Engineering (WCRE), Beverly, USA*, pages 305–308, 2010.

[31] ISO. ISO:26262-6:2011 Road vehicles - functional safety - part 6: Product development at the software level, 2011.

[32] H. Jonsson, S. Larsson, and S. Punnekkat. Agile practices in regulated railway software development. In *Proc. of the 23rd IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), Dallas, Texas*, pages 355–360. IEEE, 2012.

[33] V. Katta and T. Stalhane. A conceptual model of traceability for safety systems. In *CSDM-Poster*, 2010.

[34] M. Kifer and G. Lausen. F-logic: A higher-order language for reasoning about objects, inheritance, and scheme. In *Proc. of the ACM SIGMOD International Conference on Management of Data, Portland, USA*, pages 134–146, 1989.

[35] M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *Journal of the ACM (JACM)*, 42(4):741–843, 1995.

[36] X. Larrucea, A. Combelles, and J. Favaro. Safety-critical software [guest editors' introduction]. *IEEE Software*, 30(3):25–27, 2013.

[37] P. Mäder, O. Gotel, and I. Philippow. Getting back to basics: Promoting the use of a traceability information model in practice. In *Proc. of the 5th International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE), Vancouver, Canada*, pages 21 –25, 2009.

[38] P. Mäder, O. Gotel, and I. Philippow. Motivation matters in the traceability trenches. In *Proc. of the 17th IEEE International Conference on Requirements Engineering (RE), Atlanta, Georgia, USA*, pages 143–148, 2009.

[39] P. Mäder, P. L. Jones, Y. Zhang, and J. Cleland-Huang. Strategic traceability for safety-critical projects. *IEEE Software*, 30(3):58–66, 2013.

[40] J. C. Maxwell and A. I. Anton. A refined production rule model for aiding in regulatory compliance. Technical report, North Carolina State University. Department of Computer Science, 2010.

[41] S. Nejati, M. Sabetzadeh, D. Falessi, L. Briand, and T. Coq. A sysml-based approach to traceability management and design slicing in support of safety certification: Framework, tool support, and case studies. *Information and Software Technology*, 54(6):569–590, 2012.

[42] The Open-DO Initiative, www.open-do.org, 2013.

[43] R. K. Panesar-Walawege, M. Sabetzadeh, and L. Briand. A model-driven engineering approach to support the verification of compliance to safety standards. In *Proc. of the 22nd IEEE International Symposium on Software Reliability Engineering (ISSRE), Hiroshima, Japan*, pages 30–39, 2011.

[44] M.-A. Peraldi-Frati and A. Albinet. Requirement traceability in safety critical systems. In *Proc. of the 1st Workshop on Critical Automotive applications: Robustness & Safety, CARS'10, Valencia, Spain*, pages 11–14, 2010.

[45] B. Ramesh. Factors influencing requirements traceability practice. *Communications of the ACM*, 41(12):37 – 44, 1998.

[46] B. Ramesh and M. Jarke. Toward reference models for requirements traceability. *IEEE Transactions on Software Engineering*, 27(1):58–93, 2001.

[47] Rate Adjustment by Managing Inflows (RAMI), www.chris-edwards.org/340, 2013.

[48] P. Rempel, P. Mäder, and T. Kuschke. An empirical study on project-specific traceability strategies. In *Proc. of the 21st IEEE International Requirements Engineering Conference (RE'13), Rio de Janeiro, Brasil*, pages 195–204, 2013.

[49] P. Rempel, P. Mäder, T. Kuschke, and I. Philippow. Requirements traceability across organizational boundaries - a survey and taxonomy. In *Proc. of the 19th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ'13), Essen, Germany*, pages 125–140. Springer, 2013.

[50] RTCA/EUROCAE. DO-178B/ED-12B: Software considerations in airborne systems and equipment certification, 2000.

[51] Secure Auditing for Linux (SAL), secureaudit.sourceforge.net, 2013.

[52] P. Sánchez, D. Alonso, F. Rosique, B. Álvarez, and J. A. Pastor. Introducing safety requirements traceability support in model-driven development of robotic applications. *IEEE Transactions on Computers*, 60(8):1059–1071, 2011.

[53] TOPCASED the open-source toolkit for critical systems, www.topcased.org, 2013.

[54] Flora-2, http://flora.sourceforge.net, 2013.

[55] TOPCASED-REQ, gforge.enseeiht.fr/projects/topcased-req, 2013.

[56] TOPCASED-SAM, gforge.enseeiht.fr/projects/topcased-sam, 2013.

[57] Q. Yang, J. J. Li, and D. M. Weiss. A survey of coverage-based testing tools. *The Computer Journal*, 52(5):589–597, 2009.