Reuse-Based Test Traceability: Automatic Linking of Test Cases and Requirements

Thomas Noack, Thomas Karbe
Technische Universität Berlin,
Daimler Center for Automotive IT Innovations (DCAITI)
Berlin, Germany
Email: {thomas.noack,thomas.karbe}@dcaiti.com

Steffen Helke
Brandenburg University of Technology
Cottbus - Senftenberg (BTU)
Cottbus, Germany
Email: steffen.helke@b-tu.de

***Abstract*—Safety standards demand full requirement traceability, which includes a complete tracing between requirements and test cases to stipulate how a requirement has to be verified. However, implementing such a concept rigorously is time-consuming and costly. Furthermore, in the automotive industry this cost is repeatedly incurred for each vehicle series, because in contrast to other development artefacts, reuse strategies for trace links have not yet been sufficiently researched. This paper presents the novel approach of Reuse-based Test Traceability, which allows for a more cost-effective implementation of trace links in certain cases. First, we identify and formalize a scenario, the so called RT-Problem, for reusing trace links between test cases and reused requirements, which has been observed in industry practice. Next, based on this formalization we propose a 3-layered method, which automatically creates links between test cases and reused requirements. For reasons of practicality, we focus on the first layer, which represents a transitive test-link reuse. Finally, we present the results of two field studies demonstrating that our approach is feasible in practice. As the main contribution of this work we show that the automated reuse of test cases on the basis of reused requirements is both possible and useful.**

Keywords–Reuse; Requirements; Test cases; Traceability.

I. MOTIVATION

New safety standards like ISO 26262 mean demand for traceability is higher than ever. Consequently, automotive companies must work hard to establish traceability for every phase in the V-Model. For instance, if a software error occurs, the specific part of the source code that has caused it should be identified. This is achieved by trace linking development artefacts. In hierarchical development processes, links between requirements and test cases are some of the first to arise. These links are an integral part of a relationship network. For example, an error is discovered by a test case. This test case is trace linked with a system requirement, which in turn is connected to the source code. Each kind of comprehensibility necessitates links between the requirements involved.

However, rigorously implementing such a concept is time-consuming and costly. Furthermore, in the automotive industry this cost must be paid for each vehicle series project, repeatedly, because in contrast to other development artefacts, reuse strategies that generate trace links automatically have not been sufficiently researched.

Therefore, we propose a novel method for reuse-based traceability, which extends [1] and allows for a more cost-effective implementation of trace links in certain cases.

Among other things, the ISO standard defines a demand for two categories of trace links. The first is called *Test Traceability* – ISO 26262 Pt. 8 [2, p.25] – and relates to a link convention for test specifications. Each specification in a test case must include a reference to the version of the associated work product. The second category relates to *Reuse-based Traceability* – ISO 26262 Pt. 6 [3, p.20] – which demands that every safety-related software component must be classified according to information on reuse and modification. Thus, the standard defines four classes: newly developed, reused with modification, reused without modification, and a commercial off-the-shelf product.

Our approach contributes to both claims. First, we provide a cost-effective technique for automatically generating trace links between test cases and requirements, which addresses the test traceability of the ISO standard. Secondly, we provide the generation of trace links between requirements or test cases from a previous project and the corresponding counterparts in a new project, to indicate that previous artefacts have been reused. Furthermore, our framework allows for these links to be qualified, by using types reflecting whether an artefact was modified or not.

Structure. The next section introduces a motivating example, illustrating a scenario where trace links can be reused. The section also gives important definitions of development artefacts. The section closes with the presentation of the RT-Problem (Reuse-based Test Traceability Problem), which forms the basis of this paper. Section III gives pointers to related work. We briefly survey existing traceability models and methods. We also identify limitations of these approaches to justify the need for this work. Section IV presents our 3-layered method, where we focus on the first layer, the so-called RT-linking technique. Section V describes theoretical concepts behind our RT-linking strategy. Subsequently, Section VI presents the results of two field studies demonstrating that our approach is feasible in practice. We also compare our technique with the previous manual procedure. To give the reader an impression of the complete method, Sections VII and VIII sketch the second and the third layer. The paper closes with conclusions and future work in Section IX.

II. System requirements and test cases

A. Introductory example

Figure 1 shows an example based on real specification documents. The upper left box contains artefacts from a previous vehicle series project, e.g., vehicle function *1000: Interrupt of front wiping during engine start*. Requirements *1001* and *1002* refine vehicle function *1000*. The lower box contains test cases, e.g., test case *5376: Washing during engine start*. Requirement *1002* and test case *5376* are connected via a trace link.

Requirements reuse. Reuse happens in all phases of the V-Model; therefore, reuse also applies to requirements. Requirements are reused from previous vehicle series in order to specify a new vehicle series. Technically, this reuse is achieved by copying and adapting the old requirements to the new vehicle series project. The upper right box in Figure 1 shows the reuse requirements. For example, the reuse requirement *1002-new* has been changed most: the description of the *washing interruption* has been moved.

Test trace reuse. This adaptation of requirements exemplifies the main feature of Reuse-based Test Traceability: Is test case *5376*, which already verifies source requirement *1002*, also suitable for verifying target requirement *1002-new*? More succinctly: Can test case *5376* be linked with target requirement *1002-new*?

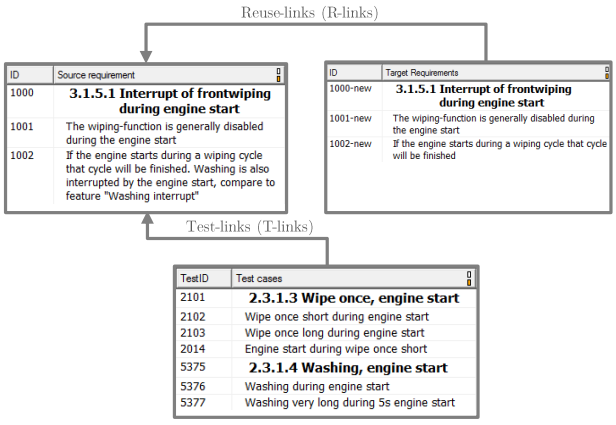


Figure 1: Example requirements and test cases in DOORS

B. System Requirements Specification (SRS)

A vehicle is described by many SRS. Every system, like the *Wiper Control* or the *Outside Light Control* is specified in one SRS. The main engineering artefacts in SRS are vehicle functions and the system requirements that refine them. Individual vehicle functions can be very extensive. For instance, the function *wipe windscreen* is characterized by several activation possibilities, wipe stages, passenger and pedestrian splash protection, etc., and therefore is refined by more than 300 requirements.

Count per vehicle series. If one function alone can have 300 requirements, how many requirements does a whole vehicle series have? The following estimate gives a vague idea: Modern vehicles have up to 100 electronic control units (ECU). Usually, multiple automotive systems run on each ECU. For simplicity, we assume that only one software system runs on each ECU. Further, we assume that midsize systems have at least 1000 requirements or more. Using these assumptions, a modern vehicle can accumulate hundreds of thousands or even millions of requirements.

Requirements classification. In addition, requirements have classifying properties such as Automotive Safety Integrity Levels (ASIL), testability, ownership, supplier status or dependencies to other SRS. Therefore, requirement categories exist, e.g., safety critical, testable or highly dependent requirements.

C. System Test Specification (STS)

Each SRS has at least one associated STS, which contains test cases to verify the correct implementation of the requirements. The structure of these test cases corresponds to the common schema: Pre-condition, post-condition, pass-condition, test steps etc. [4, p.263]. Each requirement which is classified as testable is trace linked to at least one test case. This facilitates comprehensibility to determine the requirements-based test case coverage.

Count per vehicle series. Again, the question arises: How many test cases does an entire vehicle series have? Because the test case count corresponds to the requirements count, a modern vehicle has at least as many test cases as requirements. Usually, more tests than requirements exist.

Test case classification. Like requirements, test cases have classifying properties such as test goals, test levels or test platforms. While test goals result from quality models as proposed in ISO 9126 [5], test levels describe the right branch of the V-Model. Automotive-specific test platforms include *Vehicle Network* or *HiL* (Hardware-in-the-Loop).

D. Test Concept (TC)

The ISO 26262 dictates the existence of a TC. It defines which test objects must be tested at which test level on which test platform in order to fulfil which quality goals (What? When? Where? Why?). The TC determines the relationship between the left and right branches of the V-Model. We focus on the system level of the V-Model because Reuse-base Test Traceability uses the *requirement* test object type. In our case, the TC defines which test cases must be trace linked with which requirements to sufficiently verify a vehicle series.

Usage of requirements and test case classification. The TC does not contain specific system requirements or test cases. It relies on the classifying properties of the artefacts involved. For instance, a requirement's ASIL ranking influences testing expenses because it is strongly related to the *test goal* and *test level* properties of the test case. The higher the ASIL ranking, the more test cases must be trace linked with requirements. The TC allows us to assess the trace link coverage between SRS and STS.

E. RT-*problem*

Reuse-based Test Traceability relies on a specific situation observed in industrial practice: The RT-problem. Figure 2 depicts how the RT-problem reflects the *Reuse* relationship between two SRS and the *Test* relationship between an STS and these two SRS. In the example, a requirement of the function *front wiping* (fw) from the SRS_{Src} has been reused by fw′ in the SRS_{Tgt}.

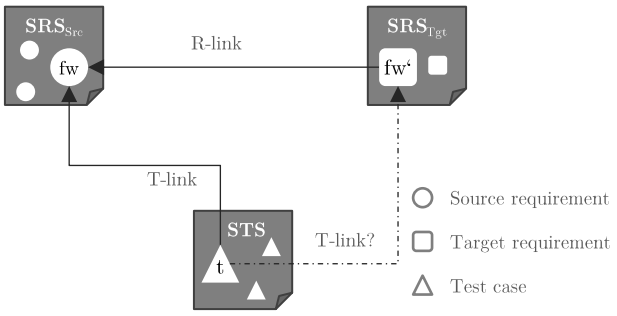


Figure 2: RT-problem within specification documents

R: Reuses. Among others, one reuse method is to copy an entire SRS into the project folder of the new vehicle series project. Thus two SRS result: the SRS_{Src} of the previous vehicle series and the adapted SRS_{Tgt} of the current vehicle series. Both SRS are in a reuse relationship: The SRS_{Tgt} reuses the SRS_{Src}.

T: Tests. Interestingly, the STS is not copied between vehicle series. Instead, the test cases are reused by redirecting trace links from the SRS_{Src} to the reusing SRS_{Tgt}. Overall, the RT-problem reflects the situation before the STS enters a test relationship with the SRS_{Tgt}.

R- and T-links. Technically, artefacts are connected by trace links. An R-link $fw' \rightarrow_R fw$ points from a target requirement fw′ to a source requirement fw. We also say (fw', fw) is a reuse pair. A T-link $t \rightarrow_T fw$ points from a test case t to a (source) requirement fw.

Solving the RT-problem. The RT-problem is unsolved if the dashed T-link in Figure 2 does not exist. In this paper, we propose transitive RT-linking, a new technique to set the T-link $t \rightarrow_T fw'$ to the target requirement fw′.

Business case. Earlier we estimated a whole vehicle series might have hundreds of thousands of requirements and test cases. Therefore, the RT-problem must be solved hundreds of thousands or even millions of times for each vehicle series project. Usually, test cases are T-linked during their creation. This T-linking requires little effort in comparison to the creation of the test case. STS are test case collections which are maintained over multiple vehicle series generations. In each new vehicle series project those STS must be linked manually. We observed that both time and motivation play an important role in why fewer and fewer T-Links exist from project to project. The primary goal of Reuse-based Test Traceability is that test cases only need to be T-linked with requirements once, at the time they are first written down.

III. Related work

A trace link connects trace artefacts and defines the type of relation between them [6, p.104]. In general, traceability is the possibility to establish and use trace links [7, p.9]. Thus, traceability enables comprehensibility. A traceability (information) model defines all possible artefacts and their types, as well as all possible links and their types [7, p.13]. Our work contributes to the special field of requirements traceability. Requirements traces are trace links between requirements and other software development artefacts [8, p.91]. Requirements traces always have a direction: forwards, backwards, inter or extra.

Link direction. A forward-trace connects a requirement with artefacts which have been created later in the development process. Examples of forward traces are links to architectural artefacts, source code or test cases. A backward-trace documents the origin of a requirement. Examples are links from laws or standards. An inter-trace links a requirement with another requirement. These links can reflect dependencies, refinement or even reuse. An extra-trace links a requirement with a non-requirement. Examples are architectural artefacts, source code or test cases.

Link direction in the RT-problem. An RT-problem consists of two links: an R-link and a T-link. The R-link connects a source and a target requirement. Therefore, it is an inter-trace link. Simultaneously, it is a backward-trace link because it reflects the origin of the target requirement. The T-link connects a test case with a source requirement. Thus, it is an extra- and forward-trace link. We observe that even though the RT-problem is very simple it contains inter, extra, forward and backward trace links.

A. *Traceability models*

Traceability models define the involved and linkable artefacts and the possible link types [9, p.106]. The RT-problem is a traceability model. The concept of traceability models appeared early in the development of software engineering: the first models appeared in the 1980s. The following paragraphs introduce relevant traceability models from the past 25 years.

General models without explicit T-links. The SODOS model [10] represents linkable artefacts via a relational database scheme. The trace links are freely configurable. Thus, SODOS is capable of connecting everything with everything. In the early 1990s, hypertext became very popular. The idea was to specify requirements and other software engineering artefacts by means of hypertext and link them using hyperlinks. Examples of hypertext traceability models include HYDRA [9], IBIS [11], REMAP [12], RETH [13], and the TOORS [14] model. Hyperlinks are mainly generic, so everything can be connected with everything else. Around the turn of the century, traceability models shifted their focus from hypertext models to UML-based traceability models [15] [16]. In accordance with the SOTA (State of the Art), older traceability models are more general while newer models are more specific. Newer work focuses on links between requirements [17] or links between requirements and design artefacts [18]. Although it is possible to define T-links in general traceability models, the models discussed here do not explicitly support T-links.

Models with T-links. In recent years, software testing has become increasingly popular. Thus, T-links have become an explicit part of traceability models. Ibrahim et al. propose the Total Traceability Model [19]. They consider requirements (R), test cases (T), design (D) and code (C). The model is exhaustive because it supports pair-wise extra-traces between the artefacts R-T, R-D, R-C, T-D, T-C and D-C. Furthermore, it supports the inter-traces D-D and C-C. Asuncion et al. have developed an End-to-End traceability model [20]. They consider marketing requirements (M), use cases (U), functional requirements (F) and test cases (T), which can be extra-linked in a M-U, U-F and F-T pipeline. Kirova et al. propose a traceability model, which uses performance requirements (PR), high level requirements (HLR), architectural requirements (AR), system requirements (SR), high level design (HLD), low level design (LLD), test cases (T) and test plans (TP). Kirova's model allows links between PR-AR, PR-SR, PR-T, HLR-SR, AR-SR, AR-T, AR-TP, AR-HLD, AR-LLD, SR-T, SR-TP, SR-HLD and SR-LLD. Azri and Ibrahim propose a metamodel [21] to allow trace links between arbitrary artefacts, including code, roles and even output files from developer tools.

RT-problem. The related work commonly draws holistic traceability pictures. Thus, a standard goal is to define holistic traceability models and exhaustively list the engineering artefacts and possible trace link types. However, the RT-problem is a specialized traceability model which focuses on a narrow set of circumstances. By using T-links explicitly, it focuses on the relationship between requirements and test cases. Additionally, the RT-problem introduces a new factor, the representation of requirements reuse with the help of R-links.

B. Traceability methods and techniques

The Requirements Traceability Matrix (RTM) was one of the first techniques that could systematically handle traceability [22]. The RTM is simply a table of requirement rows and linkable artefact columns. Each cell in the table represents a possible link. Requirements engineering tools like DOORS [23] support the RTM. Newer work is based on the idea of automatically creating trace links between artefacts and providing impact analysis. Two surveys [24, p.2] [25, p.31] categorize the SOTA of traceability methods as follows: event-based, rule-based, feature model-based, value-based, scenario-based, goal-based and information retrieval-based.

Event-based Traceability (EBT). EBT [26] introduces an event service where any linkable artefacts are registered. The service takes over the traceability and artefacts are no longer linked directly. Thus, EBT supports maintainability, as events trigger when artefacts change.

Rule-based Traceability (RBT). RBT [27] applies grammatical and lexical rules to find artefacts in structured specification documents and use case diagrams. A pairwise rule matching algorithm looks for artefacts, which match a rule. RBT links those related artefacts.

Feature Model-based Traceability (FBT). FBT [28] uses the feature as a connecting element between requirements and architecture as well as requirements and design. FBT uses several consistency criteria, e.g., whether each feature has at least one requirement and test case.

Value-based Traceability (VBT). VBT [29] assumes that a complete linkage between all involved artefacts is not feasible. Thus, VBT supports prioritized requirements and differently precise traceability schemes. The goal of VBT is to distinguish between links that generate benefits and links that only produce costs.

Scenario-based Traceability (SBT). SBT [30] uses scenarios, such as state chart paths, which are linked with requirements and code fragments. The creation of new links is performed transitively via code analysis. SBT is capable of completing links between requirements and code.

Goal-based Traceability (GBT). GBT [31] uses a quality model to define nonfunctional quality goals. Those nonfunctional goals are then connected to functional requirements. The goal of GBT is to trace the change impact from functional requirements to nonfunctional goals.

Information Retrieval-based Traceability (IBT). In recent years, IBT has become increasingly popular. Several approaches exist to finding and linking related artefacts [32]. The general idea behind IBT is to use information retrieval algorithms and similarity measures.

C. Alignment with SOTA (State-of-the-Art)

EBT propagates requirements change to linked artefacts. Thus, EBT is a technical event-based method to avoid manual tracing of impact and manual button clicks. Although it would be interesting to automatically execute the RT-problem solving technique after requirements reuse, we do not need EBT to solve the RT-problem methodically. RBT uses grammatical and lexical analysis to find similar requirements. Although it would be interesting to find reuse pairs with RBT, we assume that all R-links exist. Because of the importance of variability in the automotive domain, other research focuses on FBT [33]. VBT tries to reduce the number of links in order to reduce maintenance costs. The RT-problem is a specialized traceability model to reflect a simple circumstance and analyze it holistically. Thus, we do not want to remove any links in this way. SBT introduces an additional connecting artefact to link requirements and code. Because we focus on requirements and test cases, we do not desire new artefacts. Thus, SBT is also not suitable to solve the RT-problem.

New technique: RT-linking. Firstly, we propose a new technique to transitively link test cases by considering requirements reuse: RT-linking. We present a 3-layered method which integrates our RT-linking with parts of the SOTA. RT-linking is the primary method layer for creating new trace links between test cases and reusing requirements completely. To make the linking more precise we need additional filtering techniques, which are executed during the complete RT-linking. These filtering techniques define the other two layers of the 3-layered method. While the first layer establishes all links between test cases and reusing target requirements, the second and third layers filter out or highlight suspicious linking situations. The second layer uses the idea behind GBT to assess the test coverage with respect to test goals and other testing criteria. The third layer adapts IBT to search for similar RT-problems.

IV. 3-LAYERED METHOD

Figure 3 depicts the method layers as first proposed in [1]. Each layer consists of the same three phases. The specific tasks in each phase differ depending on the characteristics of the layer. Each subsequent layer enhances its predecessor's phases via additional tasks. The general tasks performed in the three phases are as follows:

- *Extract* RT-problems from SRS_{Src}, SRS_{Tgt} and STS.
- *Set/Filter* T-links from STS to SRS_{Tgt}.
- *Assess* T-links and highlight the link status.

Figure 4 depicts the 3-layered method in more detail. We will use the depicted example to briefly introduce the layers.

A. First layer: Transitive RT-Linking

Extract RT-problems. Figure 4 depicts an RT-problem: The target front wiping fw_{Tgt} reuses the source front wiping fw_{Src}. Thus, (fw_{Tgt}, fw_{Src}) is a reuse pair. The test case fw_{Test} has a T-link to fw_{Src}. Because fw_{Test} is not yet T-linked to fw_{Tgt}, one RT-problem is extracted.

Set T-links. The T-links of all extracted RT-problems are set transitively: If a test case is T-linked with a source requirement and this source requirement is R-linked with a target requirement, then the test case is also T-linked with the target requirement.

Assess T-links. Two scenarios can occur for each RT-linked target requirement: (a) It is textually identical to the source requirement and hence a T-link needs no review or (b) it has been changed and, therefore, the T-link must be reviewed manually. The third phase of each layer highlights the SRS_{Tgt} according to the assessment results.

B. Second layer: Test Concept-driven filtering

Extract RT-problems. Figure 4 indicates that the test case fw_{Test} meets the test goal *correctness of interfaces*. This information is extracted from the STS and appended to the test case of the RT-problem. We say that the RT-problem is augmented with the classifying property *test goal*.

Filter T-links. The TC defines whether a test case is needed to sufficiently verify a vehicle series. In Figure 4, the TC defines that the test goal *correctness of interfaces* must be met. Because the TC demands for a interface test case such as fw_{Test}, the RT-linking connects fw_{Test} and fw_{Tgt}. Otherwise, fw_{Test} would have been filtered out.

Assess T-links. While the first layer can only make statements about the existence of T-links, the second layer also considers the TC. Therefore, for each target requirement the following more detailed scenarios arise: (a) missing/superfluous test goals, (b) missing/superfluous test levels and (b) missing/superfluous test platforms. The SRS_{Tgt} is highlighted according to the TC coverage. TC-driven filtering has been proposed in [34].

C. Third layer: Case-Based filtering

Extract RT-problems. We assume that the requirements in Figure 4 have a classifying property, *interfaces*. While the source requirement fw_{Src} has an interface to the *column switch*, the target requirement has an additional interface to the *rain sensor*. In other words, the interfaces of the reuse pair (fw_{Tgt}, fw_{Src}) have changed. Again, we use the classifying properties to augment the extracted RT-problem.

Filter T-links. A Case Base contains RT-cases. RT-cases are RT-problems which include an RT-decision and a review note. The RT-decision defines whether a T-link can be set to a target requirement. Figure 4 depicts a simple RT-case: if the interfaces change, interface tests must be reviewed. This RT-case is very similar to our current RT-problem. Thus, the RT-decision *Review needed* is used to solve it.

Assess T-links. While the second layer only uses classifying test case properties, the third layer relies on fully augmented RT-problems. That means RT-cases can be defined freely by taking any classifying property into account. Thus, the assessment scenarios are as numerous as the RT-case possibilities. Finally, the RT-cases' review notes are copied into the SRS_{Tgt}.

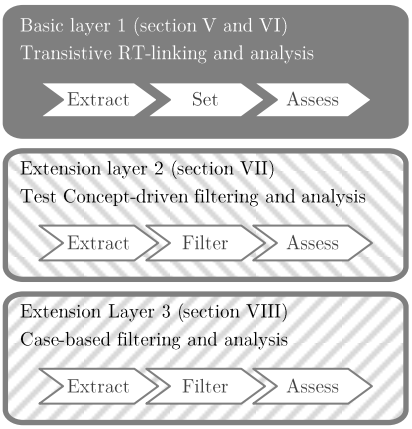


Figure 3: 3-layered method

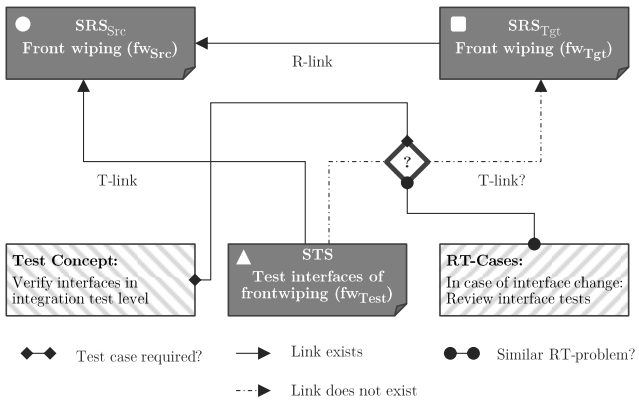


Figure 4: 3-layered method in more detail

V. RT-LINKING: CONCEPTS

A. Fundamental Example

Figure 1 showed an example with real requirements and test cases. However, real requirements and test cases are very extensive and also confidential. Therefore, the following sections will use the abstract example in Figure 5 to simplify the problem and concentrate on its relevant features.

Phase 1: Extract RT-Problems. Figure 5 shows the specification documents SRS_{Src}, STS and SRS_{Tgt}. Each example SRS contains one vehicle function, which is refined by requirements. While SRS_{Src} contains the source requirements src_i, SRS_{Tgt} contains the target requirements tgt_i. The *Reuse* column indicates for both SRS which source or target requirements the R-links point to. The *Test* column in SRS_{Src} indicates whether a source requirement is connected via a T-link with a test case. The *T-links: Before* column in STS displays which requirements the test cases were linked with before RT-linked them. Figure 5 contains three RT-problems with the following R- ($\rightarrow_R$) links and T-links ($\rightarrow_T$):

- $tgt_1 \rightarrow_R src_1$ and $test_1 \rightarrow_T src_1$,
- $tgt_2 \rightarrow_R src_2$ and $test_2 \rightarrow_T src_2$,
- $tgt_3 \rightarrow_R src_3$ and $test_{3/4} \rightarrow_T src_3$.

Phase 2: Execute RT-linking. The transitive RT-linking uses the following assumption:

IF	a target requirement reuses a source requirement
AND IF	a test case verifies a source requirement
THEN	the test case also verifies the target requirement.

After performing the RT-linking, the *T-Links: After* column contains the names of the source and target requirements which are linked with the test case by a T-Link. The *T?* column in SRS_{Tgt} indicates whether there is a T-link to a test case. The column is set to *Check* if the text for the target requirement has changed or if RT-inconsistencies (see **Phase 3**) have been uncovered. As Figure 5 shows, the second layer of the transitive RT-linking results in three solved RT-problems:

- $tgt_1 \rightarrow_R src_1$ and $test_1 \rightarrow_T src_1 \Rightarrow test_1 \rightarrow_T tgt_1$,
- $tgt_2 \rightarrow_R src_2$ and $test_2 \rightarrow_T src_2 \Rightarrow test_2 \rightarrow_T tgt_2$,
- $tgt_3 \rightarrow_R src_3$ and $test_{3/4} \rightarrow_T src_3 \Rightarrow test_{3/4} \rightarrow_T tgt_3$.

Phase 3: Assess T-links. The *similarity* column of SRS_{Tgt} represents the textual similarity of a reuse pair in percent. It is calculated with the help of well-known similarity measures, e.g., Dice, Jaro-Winkler or the Levenshtein distance [35]. As well as textual similarity, several other inconsistencies might occur, e.g., the SRS_{Src} describes a front and a rear wiper. A test case verifies both source requirements: If the reverse gear is engaged and if the column switch is pushed then the front and rear wipers will wipe. Now the target vehicle series does not provide a rear wiper. However, the SRS_{Tgt} reuses the front wiper requirement and the T-link from the test case. The inconsistency arises because the test case verifies a functionality (rear wiping) which does not exist in the target vehicle series.

Source System Requirement Specification (SRS_{Src})

Source requirements	Reuse	Test
1 Function Src		
src 1	tgt 1	Yes
src 2	tgt 2	Yes
src 3	tgt 3	Yes
src 4: will be discarded		Yes

System Test Specification (STS)

Test cases	T-links: before	T-links: after
1 Tests		
test 1	src 1	src 1, tgt 1
test 2	src 2	src 2, tgt 2
test 3/4	src 3 src 4	src 3, tgt 3 src 4

Target System Requirement Specification (SRS_{Tgt})

Target requirements	Reuse	Test	Similarity	Incons.	Review note
1 Function Tgt					
tgt 1: the same	src 1	Yes	100		
tgt 2: almost the same	src 2	Check	80		- Textual change
tgt 3: not the same	src 3	Check	60	II_Src	- Textual change - Test 3/4 links to Src 4 (discarded)
tgt 5: new		No			

Figure 5: Fundamental Before-After example

B. RT-*Linking*

The following basic sets describe the RT-problem:

$$SRS_{Src} \hat{=} \text{set of source system requirements}$$
$$SRS_{Tgt} \hat{=} \text{set of target system requirements}$$
$$STS \hat{=} \text{set of system test cases}$$

The upper left circle in Figure 6 represents SRS_{Src}, i.e., the set of all source system requirements. The upper right circle analogously represents SRS_{Tgt}, i.e., the set of all target system requirements. The lower circle shows the STS, i.e., the set of all system test cases. All three sets are disjoint, thus the overlapping areas in Figure 6 do not symbolize intersected sets, but linked elements between the disjoint sets.

Reuse pairs: R-links between requirements. An R-link $r_{tgt} \rightarrow_R r_{src}$ always points from target to source. A reuse target requirement from SRS_{Tgt} therefore points towards a reused source requirement from SRS_{Src}. Both target and source requirements can have multiple outgoing or incoming R-links. The following sets describe this information:

$$R := \{(r_{tgt}, r_{src}) \in SRS_{Tgt} \times SRS_{Src} \mid r_{tgt} \rightarrow_R r_{src}\}$$
$$R_{R,Src} := \{r_{src} \in SRS_{Src} \mid \exists r_{tgt} : (r_{tgt}, r_{src}) \in R\}$$
$$R_{R,Tgt} := \{r_{tgt} \in SRS_{Tgt} \mid \exists r_{src} : (r_{tgt}, r_{src}) \in R\}$$

The set R contains reuse pairs (r_{tgt}, r_{src}) which show that a R-link is pointing from r_{tgt} to r_{src}. Thus, $(r_{tgt}, r_{src}) \in R$ is a synonym for $r_{tgt} \rightarrow_R r_{src}$. The pair (r_{tgt}, r_{src}) is also a reuse pair. The sets $R_{R,Src}$ and $R_{R,Tgt}$ contain all requirements that are part of a reuse pair. Thus, $R_{R,Src}$ contains all reused source requirements from SRS_{Src}, and $R_{R,Tgt}$ contains all reusing target requirements from SRS_{Tgt}.

T-links from Test Cases to Requirements. A T-link $t \rightarrow_T r$ points from a test case to a requirement. An *active* test case points towards at least one requirement. A *verified* requirement has at least one incoming T-link from a test case. A test case can verify multiple source and/or target requirements, depending on whether it points into SRS_{Src}, into SRS_{Tgt} or both.

$$
\begin{aligned}
T &: SRS_{Src} \cup SRS_{Tgt} \rightarrow \mathcal{P}(STS) \\
T(r) &:= \{t \in STS \mid t \rightarrow_T r\} \\
R_{T,Src} &:= \{r_{src} \in SRS_{Src} \mid \exists t \in STS : t \rightarrow_T r_{src}\} \\
R_{T,Tgt} &:= \{r_{tgt} \in SRS_{Tgt} \mid \exists t \in STS : t \rightarrow_T r_{tgt}\}
\end{aligned}
$$

For a given requirement r (source or target), the function T derives all test cases that are linked with it. The set $R_{T,Src}$ contains all source requirements r_{src} from SRS_{Src} for which at least one T-link $t \rightarrow_T r_{src}$ points from a test case t from STS to r_{src}. Analogously, the set $R_{T,Tgt}$ contains all target requirements r_{tgt} that are linked with at least one test case t.

RT-linking. With these formal concepts we can reformulate the assumption for the transitive RT-linking:

IF	a target requirement r_{tgt} and
	a source requirement r_{src} are linked by an R-link
AND IF	a test case t is linked with r_{src} by a T-link
THEN	t can also be linked to r_{tgt} by a T-link.

This is represented by the formula

$$r_{tgt} \rightarrow_R r_{src} \wedge t \rightarrow_T r_{src} \Rightarrow t \rightarrow_T r_{tgt}$$

An RT-link, i.e., a solution for one of the three RT-problems from the abstract example in Figure 5 is shown here:

$$tgt_1 \rightarrow_R src_1 \text{ and } test_1 \rightarrow_T src_1 \Rightarrow test_1 \rightarrow_T tgt_1$$

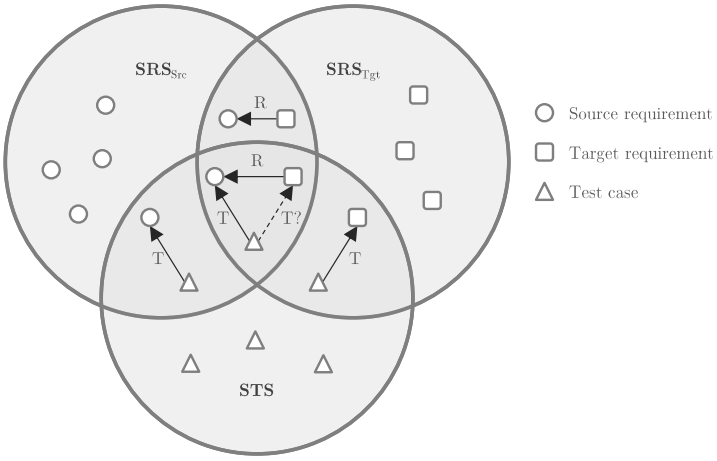


Figure 6: Each RT-problem is a RT-instance among others

C. RT-*diagram*

The RT-diagram is a means to represent the number of linking situations between requirements and test cases. It categorizes these situations into different types, e.g., reused but not tested requirements, reused and tested requirements. Figure 7 show the diagram schematically. The different segments of the diagram represent the different types of linking situations, which result from the existence or non-existence of links between the three basic sets SRS_{Src}, STS, and SRS_{Tgt}. Each segment is labelled with a different symbol (e.g., ○, □) that represents the type of the segment. In the industrial use and field studies in Section VI, the diagram segments show the number of linking situations.

RT-instances and RT-types. From now on, we call the different linking situations RT-*instances*. Every RT-instance is assigned to an RT-*type* (e.g., not tested but reused requirement). All RT-instances from the same segment of the diagram are also from the same RT-type. An RT-instance is denoted by a set, which contains either

- one artefact (meaning that this artefact is not linked)
  Corresponding types: ○, □, △,
- one link (meaning that the two linked artefacts are not linked to a third artefact)
  Corresponding types: ⊐, ⋒, ⋓,
- two links (meaning that one R-link and one T-link exists, but one T-link is missing to one of the partners of the reuse pair)
  Corresponding types: $\bullet_{inconsistent}$, or
- three links (fully linked, a solved RT-problem with a reuse pair and T-links to both partners of the pair)
  Corresponding types: $\bullet_{consistent}$.

On the next page we will examine the segments of the RT-diagram. Each segment will also be given a short name for future reference.

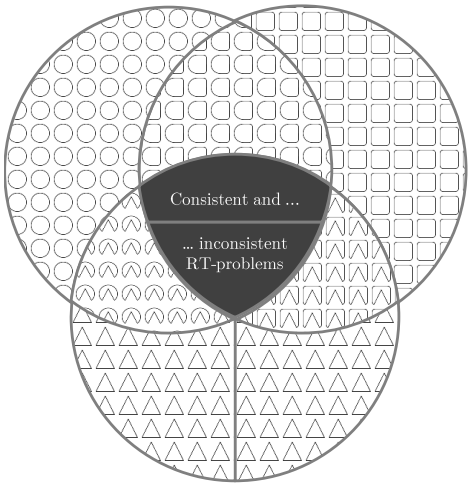


Figure 7: Types of RT-instances

Requirements without RT ($\bigcirc, \square$)**.** Figure 8a depicts those requirements which have no R-link and no T-link. The segment on the left-hand side represents all *discarded* SRS_{Src} requirements which have not been tested ($\bigcirc$). The segment on the right-hand side contains all *new* SRS_{Tgt} requirements which have no associated test cases ($\square$).

$$\bigcirc : \overline{R_{Src}}\,\overline{T_{Src}} := \{\{r_{src}\} \mid r_{src} \in SRS_{Src} \setminus (R_{T,Src} \cup R_{R,Src})\}$$
$$\square : \overline{R_{Tgt}}\,\overline{T_{Tgt}} := \{\{r_{tgt}\} \mid r_{tgt} \in SRS_{Tgt} \setminus (R_{T,Tgt} \cup R_{R,Tgt})\}$$

Reused requirements without T ($\square\!\!\square$)**.** Figure 8b depicts all reuse pairs (r_{tgt}, r_{src}), which have no T-link to either r_{tgt} or r_{src} . From the perspective of the SRS_{Tgt} these are all reusable and untested target requirements. Although we use the word *untested*, requirements with missing T-links do not remain untested in industrial practice. The mapping between requirements and test cases is then performed by engineers in an experience-based fashion. Of course, in this case testing takes place in SRS_{Tgt}; it is simply less traceable.

$$\square\!\!\square : R\,\overline{T_{Src*Tgt}} :=$$
$$\{\{r_{tgt} \rightarrow_R r_{src}\} \mid (r_{tgt}, r_{src}) \in R \wedge T(r_{tgt}) \cup T(r_{src}) = \emptyset\}$$

Tested requirements without R ($\mathbin{\reflectbox{$\ddots$}}, \ddots$)**.** Figure 8c shows all requirements that are not in a reuse relationship but which have associated test cases. The segment on the left-hand side depicts source requirements which are not reused, have no R-link but do have a T-link from a test case ($\mathbin{\reflectbox{$\ddots$}}$). The right-hand side shows all new target requirements which have no R-link but a new T-link ($\ddots$).

$$\mathbin{\reflectbox{$\ddots$}} : \overline{R_{Src}}\,T_{Src} := \{\{r_{src}\} \mid r_{src} \in R_{T,Src} \setminus R_{R,Src}\}$$
$$\ddots : \overline{R_{Tgt}}\,T_{Tgt} := \{\{r_{tgt}\} \mid r_{tgt} \in R_{T,Tgt} \setminus R_{R,Tgt}\}$$

RT-problems ($\blacktriangledown$)**.** Figure 8d shows the center of the RT-diagram. It represents the RT-problems, i.e., all RT-instances, which have a reuse pair (r_{tgt}, r_{src}) and a test case which is T-linked to at least one of the reuse partners. Therefore, the diagram center bundles three RT-types: RT-instances which have reuse pairs (r_{tgt}, r_{src}) and a T-link to r_{src} ($\blacktriangledown_{Src}$). RT-instances with reuse pairs which are only r_{tgt} T-linked ($\blacktriangledown_{Tgt}$). RT-instances which have reuse pairs and a T-link to both partners of the pair ($\blacktriangledown_{Both}$).

$$\blacktriangledown : R\,T_{Src+Tgt} :=$$
$$\left\{ \begin{array}{l} \{r_{tgt} \rightarrow_R r_{src}, t \rightarrow_T r_{src}\} \mid \\ (r_{tgt}, r_{src}) \in R \wedge t \in T(r_{src}) \wedge t \notin T(r_{tgt}) \end{array} \right\}$$
$$\cup \left\{ \begin{array}{l} \{r_{tgt} \rightarrow_R r_{src}, t \rightarrow_T r_{tgt}\} \mid \\ (r_{tgt}, r_{src}) \in R \wedge t \in T(r_{tgt}) \wedge t \notin T(r_{src}) \end{array} \right\}$$
$$\cup \left\{ \begin{array}{l} \{r_{tgt} \rightarrow_R r_{src}, t \rightarrow_T r_{src}, t \rightarrow_T r_{tgt}\} \mid \\ (r_{tgt}, r_{src}) \in R \wedge t \in T(r_{src}) \wedge t \in T(r_{tgt}) \end{array} \right\}$$

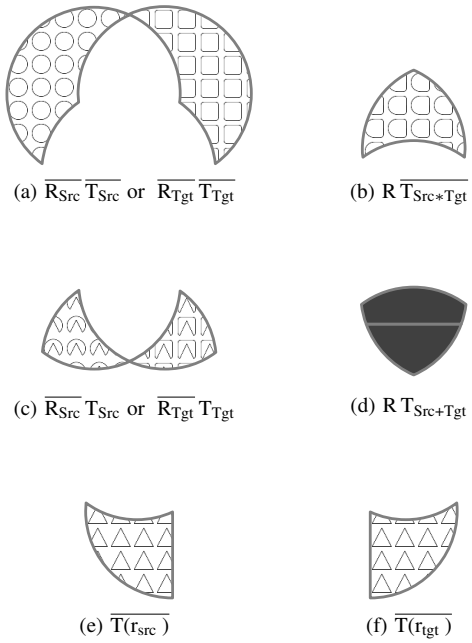
Test cases without T ($\Delta_{Src}, \Delta_{Tgt}$)**.** Figures 8e and 8f depict the test cases which have no T-links into SRS_{Src} or SRS_{Tgt}. In the context of the corresponding SRS those test cases are inactive.

$$\Delta_{Src} : \overline{T(r_{src})} := \{\{t\} \mid t \notin \bigcup_{r_{src} \in SRS_{Src}} T(r_{src})\}$$
$$\Delta_{Tgt} : \overline{T(r_{tgt})} := \{\{t\} \mid t \notin \bigcup_{r_{tgt} \in SRS_{Tgt}} T(r_{tgt})\}$$

Set of all RT-instances. An RT-instance represents a concrete link between one, two, or three artefacts. The set of all possible RT-instances RT_{Inst} is defined by:

$$RT_{Inst} := \bigcirc \cup \square \cup \square\!\!\square \cup \mathbin{\reflectbox{$\ddots$}} \cup \ddots \cup \blacktriangledown \cup \Delta_{Src} \cup \Delta_{Tgt}$$

The one-artefact instances $\{r_{src}\} \in \bigcirc$, $\{r_{tgt}\} \in \square$, $\{t\} \in \Delta_{Src}$ and $\{t\} \in \Delta_{Tgt}$ symbolize artefacts which are not linked to any other artefacts. The one-link instances $\{r_{tgt} \rightarrow_R r_{src}\} \in \square\!\!\square$, $\{t \rightarrow_T r_{src}\} \in \mathbin{\reflectbox{$\ddots$}}$, and $\{t \rightarrow_T r_{tgt}\} \in \ddots$ represent a link between exactly two artefacts, which both are not linked to any other artefact. The two-link instances $\{r_{tgt} \rightarrow_R r_{src}, t \rightarrow_T r_{src}\} \in \blacktriangledown_{Src}$, and $\{r_{tgt} \rightarrow_R r_{src}, t \rightarrow_T r_{tgt}\} \in \blacktriangledown_{Tgt}$ represent a source and a target requirement linked by an R-link and a test case which is linked to either the source requirement or the target requirement. However, the second test link is missing. The three-link instances $\{r_{tgt} \rightarrow_R r_{src}, t \rightarrow_T r_{src}, t \rightarrow_T r_{tgt}\} \in \blacktriangledown_{Src+Tgt}$ represent fully linked instances.



(a) $\overline{R_{Src}}\,\overline{T_{Src}}$ or $\overline{R_{Tgt}}\,\overline{T_{Tgt}}$ (b) $R\,\overline{T_{Src*Tgt}}$ (c) $\overline{R_{Src}}\,T_{Src}$ or $\overline{R_{Tgt}}\,T_{Tgt}$ (d) $R\,T_{Src+Tgt}$ (e) $\overline{T(r_{src})}$ (f) $\overline{T(r_{tgt})}$

Figure 8: Segments in the RT-diagram

D. RT-*inconsistencies*

We will now examine instances of the type $\bullet : R\, T_{Src+Tgt}$ in more detail. In these instances, test cases are T-linked with one or both partners of an RT-problem's reuse pair (r_{tgt}, r_{src}). But, in practice test cases are not only linked with one reuse pair. Usually, they have links to multiple reuse pairs or even requirements which have no reuse relationship to other requirements, which causes inconsistencies. Therefore, RT-inconsistencies are caused by missing T-links or unfavourable overlaps of reuse pairs with other requirements which are not part of a reuse pair, or by combining or splitting requirements. An RT-inconsistency is not necessarily an error, but an engineer should look into the RT-instance to check it. The following formulas describe the conditions for a reuse pair to be called *consistent*:

Consistency rule I. The first consistency rule says that a T-link must point to both partners in a reuse pair. If this rule is broken it indicates either overlooked source T-links or new target T-links. Two forms of rule I exist:

$$I_{Src}(r_{tgt}, r_{src}) := T(r_{src}) \subseteq T(r_{tgt})$$
$$I_{Tgt}(r_{tgt}, r_{src}) := T(r_{tgt}) \subseteq T(r_{src})$$

The first form I_{Src} says that the set of all test cases which are T-linked with the source requirement r_{src} of reuse pair (r_{tgt}, r_{src}) must also be T-linked with the target requirement r_{tgt}. The second inconsistency form I_{Tgt} is analogously defined: all test cases which are T-linked with r_{tgt} of a reuse pair (r_{tgt}, r_{src}) must also be T-linked with r_{src}.

Consistency rule II. While the first inconsistency rule assesses a reuse pair locally, the second inconsistency rule takes other requirements into account. It says that each test case that is T-linked to a given reuse pair is not allowed to test other requirements which are not part of another reuse pair. Therefore, this second rule highlights discarded source or newly added target functionality from the testing perspective. Again, two forms of inconsistency rule II exist:

$$\begin{aligned} II_{Src}(r_{tgt}, r_{src}) := \forall t \in T(r_{tgt}) \cup T(r_{src}) : \\ \forall r'_{src} \in SRS_{Src} : r'_{src} \neq r_{src} \Rightarrow \\ (t \rightarrow_T r'_{src} \Rightarrow \exists r'_{tgt} \in SRS_{Tgt} : \\ t \rightarrow_T r'_{tgt} \wedge (r'_{tgt}, r'_{src}) \in R) \end{aligned}$$

$$\begin{aligned} II_{Tgt}(r_{tgt}, r_{src}) := \forall t \in T(r_{tgt}) \cup T(r_{src}) : \\ \forall r'_{tgt} \in SRS_{Tgt} : r'_{tgt} \neq r_{tgt} \Rightarrow \\ (t \rightarrow_T r'_{tgt} \Rightarrow \exists r'_{src} \in SRS_{Src} : \\ t \rightarrow_T r'_{src} \wedge (r'_{tgt}, r'_{src}) \in R \end{aligned}$$

The first form II_{Src} says that a reuse pair (r_{tgt}, r_{src}) is consistent if all test cases of r_{src} are only T-linked with other source requirements r'_{src} which are part of a reuse pair (r'_{tgt}, r'_{src}), for some target requirement r'_{tgt}. In addition, all such test cases must be T-linked with r'_{tgt}. To improve the reade'rs understanding of inconsistency II_{Src} we will repeat the example of front and rear wipers. The front wiping requirement has been reused and the reuse pair $(front_{tgt}, front_{src})$ exists. Because the test case t is T-linked with $front_{src}$ it has been transitively RT-linked with $front_{tgt}$. Thus, t is T-linked with both partners of the front wiping reuse pair. Now, t is also T-linked with the source rear wiping requirement $rear_{src}$, which has not been reused because the new vehicle series does not provide any rear wiping. Since no reuse partner exists for $rear_{src}$, t causes $(front_{tgt}, front_{src})$ to be inconsistent II_{Src}: Test case t verifies functionality which does not exist in the target vehicle series. The second form II_{Tgt} is defined analogously from the perspective of the SRS_{Tgt}.

Consistency rule III. The third consistency rule indicates whether a target requirement has been amalgamated from multiple source requirements or a source requirement has been split into multiple target requirements. A T-link becomes problematic if a source requirement has been split. It is then unclear which target requirement needs to be T-linked. Again, two forms of consistency rule III exist:

$$\begin{aligned} III_{Src}(r_{tgt}, r_{src}) := \nexists r'_{tgt} \in SRS_{Tgt} : \\ r_{tgt} \neq r'_{tgt} \wedge (r'_{tgt}, r_{src}) \in R \end{aligned}$$

$$\begin{aligned} III_{Tgt}(r_{tgt}, r_{src}) := \nexists r'_{src} \in SRS_{Src} : \\ r_{src} \neq r'_{src} \wedge (r_{tgt}, r'_{src}) \in R \end{aligned}$$

The first form III_{Src} says that the source requirement r_{src} of the reuse pair (r_{tgt}, r_{src}) must not be a source partner of another reuse pair. III_{Tgt} says that the target requirement r_{tgt} of the reuse pair (r_{tgt}, r_{src}) must not be a target partner of another reuse pair.

Extension of the centre of the RT-diagram. The centre of the RT-diagram counts consistencies and inconsistencies. While consistently solved RT-problems are counted in the upper region of the centre, the inconsistent RT-instances are counted in the lower region. As depicted in Figure 9, we further differentiate between source and target inconsistencies in this lower centre region. Source inconsistencies X_{Src} indicate missing T-links which pointed to source requirements but not reusing target requirements or splits in source requirements. Target inconsistencies X_{Tgt} indicate progress because of newly added T-links. Source inconsistencies are bad inconsistencies and target inconsistencies are good inconsistencies.

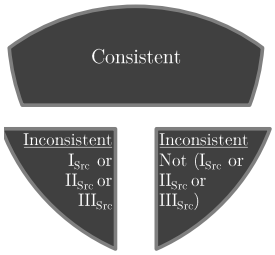


Figure 9: Inconsistencies in the diagram centre

Inconsistency I_{Src}. Figure 10a depicts the reuse pair (r_{tgt}, r_{src}) and the dashed T-link from test case t to r_{src}. The dashed T-link causes the reuse pair to be inconsistent I_{Src} because $T(r_{src}) \not\subseteq T(r_{src})$. This situation represents a typical RT-problem that needs do be solved.

Inconsistency I_{Tgt}. In Figure 10a, the reuse pair (r_{tgt}, r_{src}) would be inconsistent I_{Tgt} if the dashed T-link pointed from t to r_{tgt} instead of r_{src}, i.e., $t \rightarrow_T r_{tgt}$. In industrial practice, this situation usually occurs when a new test case has been added after reusing a requirement.

Inconsistency II_{Src}. Figure 10b depicts inconsistency II_{Src} in reuse pair (r_{tgt}, r_{src}). We clearly see that (r_{tgt}, r_{src}) is consistent I_{Src} and I_{Tgt} because $T(r_{src}) \subseteq T(r_{tgt})$ and $T(r_{tgt}) \subseteq T(r_{src})$. However, inconsistency II_{Src} for (r_{tgt}, r_{src}) is caused by the dashed T-link from t to r'_{src}. From the perspective of the reuse pair (r_{tgt}, r_{src}), test case t is T-linked with the requirement r'_{src}, which has no reuse pair partner and thus not has been reused.

Inconsistency II_{Tgt}. The reuse pair (r_{tgt}, r_{src}) in Figure 10b is inconsistent II_{Src}. It would be inconsistent II_{Tgt} if the dashed T-link pointed from t to a newly added r'_{tgt} instead of r'_{src}, which has not been reused.

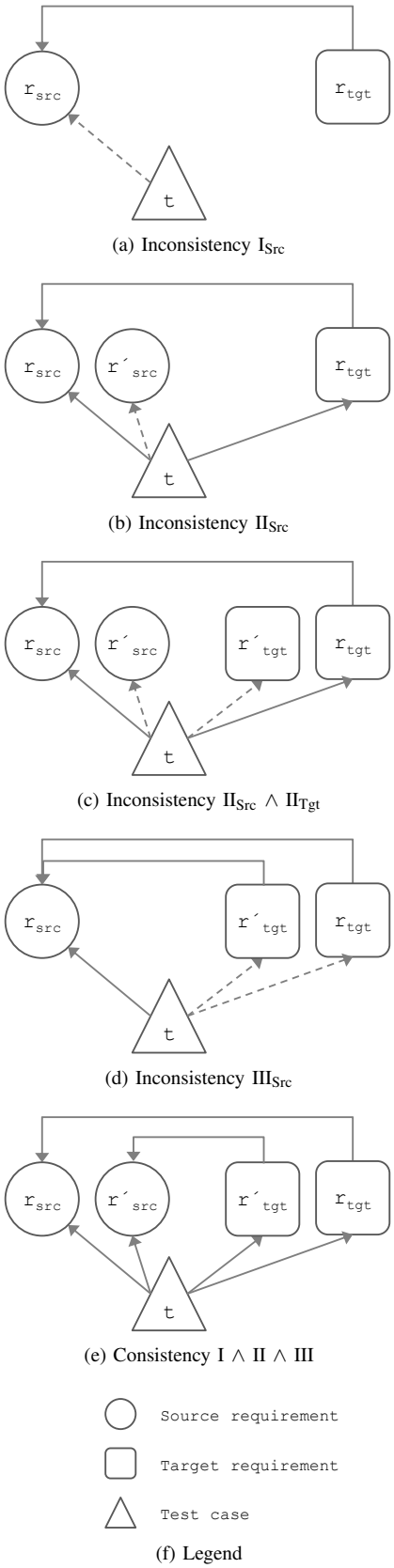
Inconsistency $II_{Src} \wedge II_{Tgt}$. Figure 10c depicts reuse pair (r_{tgt}, r_{src}), which is inconsistent II_{Src} and II_{Tgt}. Again, the reuse pair is consistent I_{Src} and I_{Tgt}. Test case t is T-linked with r'_{src} and r'_{tgt}, which are not a reuse pair. The left dashed T-link causes inconsistency II_{Src}, and the right dashed link causes inconsistency II_{Tgt}.

Inconsistency III_{Src}. Figure 10d depicts two reuse pairs, (r_{tgt}, r_{src}) and (r'_{tgt}, r_{src}), which are both consistent I_{Src} and I_{Tgt}. The source requirement r_{src} has been split into two target requirements r_{tgt} and r'_{tgt}. Test case t has been T-linked with both target requirements. Inconsistency III_{Src} occurs because source requirement r_{src} is partner of both reuse pairs.

Inconsistency III_{Tgt}. Figure 10d depicts III_{Tgt} if test case t was T-linked with all requirements of two reuse pairs, (r_{tgt}, r_{src}) and (r_{tgt}, r'_{src}).

Consistency. Figure 10e depicts two consistently T-linked reuse pairs, (r_{tgt}, r_{src}) and (r'_{tgt}, r'_{src}). Interestingly, inconsistency $II_{Src} \wedge II_{Tgt}$ from Figure 10c has been removed by adding the R-link $r'_{src} \rightarrow_R r'_{tgt}$. However, this is not a general solution, since both requirements are not necessarily partner of a reuse pair. However, this situation can still be used as an indicator to find forgotten R-links.

RT-Inconsistencies in the field. After laying the theoretical foundation for an analysis of RT-instances and RT-inconsistencies, we can now present the results of two field studies, thus showing the practical relevance of our RT-linking technique via real specification documents.



(a) Inconsistency I_{Src}

(b) Inconsistency II_{Src}

(c) Inconsistency $II_{Src} \wedge II_{Tgt}$

(d) Inconsistency III_{Src}

(e) Consistency $I \wedge II \wedge III$

(f) Legend

Figure 10: (In)Consistency examples

VI. RT-LINKING: FIELD STUDIES

From an industry perspective, the biggest advantage of the RT-linking is linking speed. However, the field studies will not focus on showing that automatic linking takes minutes instead of days (compared with manual linking). Instead, the primary goal is to show that RT-Linking is accurate.

A. Primary goal and preparation of the field study

RT-Linking is effective when it produces the same links as the current manual linking. Furthermore, the RT-linking is even more effective than the manual linking when it produces more T-links with fewer RT-inconsistencies.

Approach. In the past, RT-problems have been solved manually. We unsolve these historically solved RT-problems by removing all T-links that point to the target requirements. Next, we solve the RT-problems by RT-linking them again. The manually and automatically solved RT-problems will then be compared to assess the success of this field study.

Preparation. We were able to conduct these field studies in an industrial environment on real specification documents linked in an RT manner in IBM DOORS. These documents describe a historic linking situation between $SRS_{Src,Hist}$, $SRS_{Tgt,Hist}$ and STS_{Hist} of a past vehicle series' requirements and test reuse:

- The $SRS_{Src,Hist}$ contains source requirements.
- The $SRS_{Tgt,Hist}$ reuses $SRS_{Src,Hist}$.
- The STS_{Hist} contains test cases which point to $SRS_{Src,Hist}$ and $SRS_{Tgt,Hist}$.

The goal of the field studies is to show that automatic linking produces the same or even more T-links than the current manual linking. To achieve this, the historic documents were copied, including all links. After, all T-links between the copied STS_{RT} and the copied $SRS_{Tgt,RT}$ were removed. Thus, all documents only contained unsolved RT-problems:

- The $SRS_{Src,RT}$ is an unchanged copy of $SRS_{Src,Hist}$.
- The $SRS_{Tgt,RT}$ is a copy of $SRS_{Tgt,Hist}$. The copied SRS's have the same reuse pairs as the historical SRS's.
- The STS_{RT} is a copy of STS_{Hist} without any T-links into $SRS_{Tgt,RT}$. The T-links into $SRS_{Src,RT}$ remain.

Preparation: Special case. Before analysis of the overall linking situation can begin, we needed to reset all T-links which cause target inconsistencies. In the historic linking situation those T-links reflected new test cases which were T-linked with the $SRS_{Tgt,Hist}$ but not with the $SRS_{Src,Hist}$. We wanted the historic situation and the situation after the RT-linking to be comparable in terms of target inconsistencies. Therefore, we copied the T-links of all test cases which exclusively verified $SRS_{Tgt,Hist}$ into $SRS_{Tgt,RT}$. Those new test cases would have also been T-linked after RT-linking the artefacts from the copied documents.

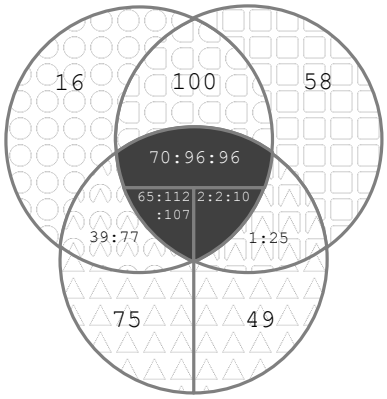
Execution. In order to compare the historic and automatic overall linking situations, all unsolved RT-problems were solved by automatically RT-linking them.

B. System A from vehicle series 1 to vehicle series 2

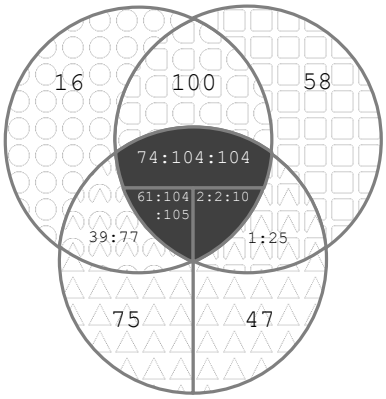
Figure 11 shows RT-diagrams to visualize the historic and RT-linking situation for a small-sized interior system.

Examination of the peripheral regions. RT-linking solves RT-problems. Because only the centre of the diagram represents RT-problems, automatic linking does not change most of the peripheral regions. Both diagrams show 16 source (○) and 58 target (□) requirements without R- and T-links. Both diagrams contain 100 reuse pairs without T-links (▢). Because the RT-linking does not change the T-links into the SRS_{Src}, both diagrams show 39 source requirements without R-links but with 77 T-links (⩓). In the field studies preparations we reset all exclusive T-links into the SRS_{Tgt}. Thus, both diagrams contain one target requirement without R-links but with 25 T-links (⩔). Both diagrams show that 75 test cases have no T-links into the SRS_{Src} ($\triangle_{Src}$). Only one peripheral region distinguishes the historic from the RT-linking situation: While 49 test cases have no T-links into $SRS_{Tgt,Hist}$, only 47 test case have no T-links into $SRS_{Tgt,RT}$ ($\triangle_{Tgt}$). This is a first clue that RT-linking is more efficient than the historic procedure.

Examination of the diagram centres. The diagram centres reveal that, historically, 70 RT-problems have been solved consistently while the RT-linking led to 74 consistently solved RT-problems. The following more detailed examination addresses this observation.



(a) Historic RT-diagram



(b) RT-diagramm after RT-linking

Figure 11: Overall linking situations

Detailed look at the inconsistencies. Table I represents the (in)consistently solved RT-problems within the diagram centres. The *Hist. situation* column represents the historic centre regions, while the *WvT linking* column stands for the centre regions of the RT-linking. The numbers R : T_{Src} : T_{Tgt} represent the count of reuse pairs (R), the count of T-links to the source requirements of the counted reuse pairs (T_{Src}) and the count of T-links to the target requirements (T_{Tgt}). Because R-links are a new concept they have been set automatically by matching document internal unique IDs of SRS_{Src} and SRS_{Tgt}. Thus, inconsistency III can not appear within the scope of the field study.

RT-diagrams and Table I. To increase the reliability of the field studies, two independent DXL scripts (DXL: Doors eXtension Language) were implemented. The first script calculates the numbers for the RT-diagram, while the second script calculates the numbers in Table I. The plausibility of the results is assured via the following rules: The numbers in the upper part of the diagram centres are the same as in row 0 of the table. The lower right region of the RT-diagram centres corresponds to the sum of the numbers in rows 2, 8 and 10. The lower left diagram centre's numbers correspond to the sum of all other table rows.

Table I: (In)consistencies in the diagram centers

(In)Consistency	Hist. situation	RT-linking
0: consistent	70:96:96	74:104:104
1: I_{Src}	1:5:4	-
2: I_{Tgt}	2:2:10	2:2:10
3: $I_{Src} \wedge I_{Tgt}$	-	-
4: II_{Src}	56:92:92	57:94:94
5: $I_{Src} \wedge II_{Src}$	4:5:0	-
6: $I_{Tgt} \wedge II_{Src}$	1:2:3	1:2:3
7: $I_{Src} \wedge I_{Tgt} \wedge II_{Src}$	-	-
8: II_{Tgt}	-	-
9: $I_{Src} \wedge II_{Tgt}$	-	-
10: $I_{Tgt} \wedge II_{Tgt}$	-	-
11: $I_{Src} \wedge I_{Tgt} \wedge II_{Tgt}$	-	-
12: $II_{Src} \wedge II_{Tgt}$	3:8:8	3:8:8
13: $I_{Src} \wedge II_{Src} \wedge II_{Tgt}$	-	-
14: $I_{Tgt} \wedge II_{Src} \wedge II_{Tgt}$	-	-
15: $I_{Src} \wedge I_{Tgt} \wedge II_{Src} \wedge II_{Tgt}$	-	-

RT-linking is effective. RT-linking is effective if at least the same test cases are automatically T-linked with $SRS_{Tgt,RT}$ as historically were T-linked with $SRS_{Tgt,Hist}$. First, the effectiveness is shown by the test cases not T-linked in the lower peripheral regions of both RT-diagrams. A simple for loop over all those test cases confirms that each test case which was not T-linked automatically was also not T-linked historically. Because fewer test cases have no T-links into $SRS_{Tgt,RT}$ after the RT-linking, the following statement is true: No test cases exist which have historical T-links into $SRS_{Tgt,Hist}$ but no automatically set T-links into $SRS_{Tgt,RT}$. Thus, the RT-linking is effective. A second for loop over all RT-instances in Table I confirms that each solved RT-problem in the *Hist. situation* column is also contained in the RT-*linking* column with the same or more T-links. Because at least the same T-links exist, RT-linking is effective. The existence of more T-links already indicates that RT-linking is even more effective than the current linking procedure.

RT-linking is even more effective. RT-linking is more effective than the current procedure if more test cases are T-linked and fewer RT-inconsistencies appear. A look at the diagram centres reveals that after the automatic RT-linking, 47 test cases do not point into the $SRS_{Tgt,RT}$. Thus, two fewer test cases are not T-linked in comparison to the historic linking situation. Given that the linking is effective, this leads to the conclusion that the RT-linking is even more effective than the current procedure. A further look at Table I confirms that more consistently solved RT-problems exist after the automatic linking. Row 0 of *Hist. situation* counts 70 reuse pairs. The source and target requirements of those 70 reuse pairs each count 96 consistent T-links. After the automatic RT-linking, 74 reuse pairs were counted. The source and target requirements are connected consistently with test cases for every 104 T-links. In conclusion, RT-linking results in more T-links and less RT-inconsistencies. Thus, RT-linking is more effective than the current manual procedure.

Origin of new consistencies. Thus far the field study has revealed that the proposed method solves more RT-problems consistently. The comparison of the centres of the two RT-diagrams showed that the upper centre region shows four additional consistently solved RT-problems, while the lower left centre region shows four fewer inconsistently solved RT-problems. Row 0 of Table I validated the plausibility of the observations. Now a question arises: From where did these new consistencies originate? We answer this question in the following sections by analysing the inconsistency transitions shown in Table II.

Transition rules (TR). Each reuse pair was analysed twice: once historically and once after the RT-linking. Therefore, we know, which inconsistency a reuse pair had in both cases. This enables us to compare the inconsistency transitions of each reuse pair. In the following, we will identify which inconsistency transitions exist and give examples for each transition that occurred. First, a summary of the four inconsistency rules: Consistency remains, inconsistency I_{Src} is eliminated, inconsistency II_{Src} remains or is eliminated, inconsistencies I_{Tgt} and II_{Tgt} remain. These rules can describe all observed inconsistency transitions.

TR_1: Consistency remains. This first transition rule says that each historic RT-pair which is consistent, remains consistent after RT-linking it automatically with test cases. This was the case for all 70 consistent RT-pairs.

TR_2: Inconsistency I_{Src} is eliminated. RT-linking eliminates inconsistency I_{Src} by definition, because it transitively sets T-links between source and target requirements. Tables I and II confirm the elimination of I_{Src}, because they never contain inconsistencies with an odd number after the RT-linking (because I_{Src} denotes 2^0).

TR_3: Inconsistency II_{Src} remains or is eliminated. II_{Src} can become consistent if an inconsistency II_{Src} occurs, because an inconsistency I_{Src} occurred at another point. Otherwise II_{Src} remains. In Table II, historic inconsistencies II_{Src} remained ($4 \Rightarrow 4$, $6 \Rightarrow 6$, $12 \Rightarrow 12$) or were eliminated ($4 \Rightarrow 0$, $5 \Rightarrow 0$) by the RT-linking.

TR_4: Inconsistencies I_{Tgt} and II_{Tgt} remain. To enable a comparison between $SRS_{Tgt,Hist}$ and $SRS_{Tgt,RT}$ in preparation of the field study, all exclusive T-links into $SRS_{Tgt,Hist}$ were also set exclusively to $SRS_{Tgt,RT}$. Exclusively means that the T-link points into SRS_{Tgt} but not into SRS_{Src}. Target inconsistencies I_{Tgt} and II_{Tgt} caused by this show that new test cases have been T-linked with the SRS_{Tgt}. Because the linking of new test cases does not impact RT-linking, all target inconsistencies remain. Table II confirms this ($2 \Rightarrow 2$, $6 \Rightarrow 6$, $12 \Rightarrow 12$).

Table II: Inconsistency transitions from *Hist.* to RT

Hist. $\Rightarrow$	RT-linking	#Transitions
$0 \Rightarrow 0$	(consistent $\Rightarrow$ consistent)	70
$1 \Rightarrow 0$	($I_{Src} \Rightarrow$ consistent)	1
$2 \Rightarrow 2$	($I_{Tgt} \Rightarrow I_{Tgt}$)	2
$4 \Rightarrow 0$	($II_{Src} \Rightarrow$ consistent)	1
$4 \Rightarrow 4$	($II_{Src} \Rightarrow II_{Src}$)	55
$5 \Rightarrow 0$	($I_{Src} \wedge II_{Src} \Rightarrow$ consistent)	2
$5 \Rightarrow 4$	($I_{Src} \wedge II_{Src} \Rightarrow II_{Src}$)	2
$6 \Rightarrow 6$	($I_{Tgt} \wedge II_{Src} \Rightarrow I_{Tgt} \wedge II_{Src}$)	1
$12 \Rightarrow 12$	($II_{Src} \wedge II_{Tgt} \Rightarrow II_{Src} \wedge II_{Tgt}$)	3

Transition examples. To improve the reader's understanding of inconsistency transitions, an example will be presented for each row in Table II. The following figures show real but anonymised transitions. Therefore, some inconsistency transitions appear isolated in a single example while other examples contain multiple transitions. The figure on the left hand side always represents the historically solved RT-problem(s), while the right hand figure always shows the same RT-problem(s), only automatically RT-linked.

Transition: Consistency remains ($0 \Rightarrow 0$). All source and target requirements in Figure 12 build a reuse pair: (A, a), (B, b). The test case t is T-linked with all partners of all reuse pairs. Thus, TR_1 applies and consistency remains.

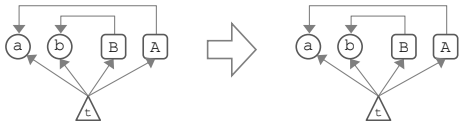


Figure 12: $0 \Rightarrow 0$

Transition: I_{Src} is eliminated ($1 \Rightarrow 0$). Figure 13 shows the reuse pair (A, a). On the left hand side the dashed T-link $t \rightarrow_T a$ causes the historic RT-problem to remain unsolved. This implies inconsistency I_{Src}. The definition of the RT-linking does not allow such situations. Rule TR_2 applies and the T-link $t \rightarrow_T A$ exists after performing the RT-linking.

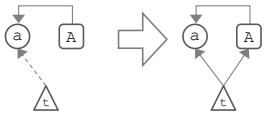


Figure 13: $1 \Rightarrow 0$

Transition: I_{Tgt} remains ($2 \Rightarrow 2$). Figure 14 depicts the reuse pair (A, a). Because $t \rightarrow_T A$ originates from the new test case t, $t \rightarrow_T a$ does not exist. This causes inconsistency I_{Tgt}, which remains according to TR_4.

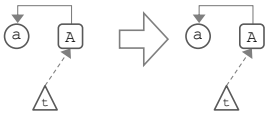


Figure 14: $2 \Rightarrow 2$

Transition: II_{Src} remains ($4 \Rightarrow 4$). Figure 15 depicts the reuse pair (B, b) and source requirement a, which has not been reused. Inconsistency II_{Src} is caused by $t \rightarrow_T a$ because t verifies functionality that does not exist on the target side. Rule TR_3 applies: II_{Src} cannot be eliminated, because it is not caused by another inconsistency I_{Src}.

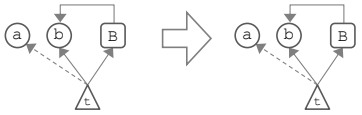


Figure 15: $4 \Rightarrow 4$

Transition: II_{Src} is eliminated ($4 \Rightarrow 0$). Figure 16 depicts several dependent RT-problems. The reuse pair (C, c) is consistent I_{Src} because t is T-linked to both partners. But the T-links t $\rightarrow_T$ a and t $\rightarrow_T$ b cause (C, c) to be inconsistent II_{Src}. Rule TR_2 applies. Thus, the inconsistencies I_{Src} of (A, a) and (B, b) are eliminated by setting t $\rightarrow_T$ A and t $\rightarrow_T$ B. Rule TR_3 applies: The pair (C, c) is now consistent because I_{Src} of (A, a) and (B, b) was the reason for its II_{Src}.

Transition: $I_{Src} \wedge II_{Src}$ becomes consistent ($5 \Rightarrow 0$). Figure 16 shows inconsistency II_{Src} is eliminated from the perspective of (C, c). Because the rules TR_2 and TR_3 also apply to the pairs (A, a) and (B, b) their inconsistencies $I_{Src} \wedge II_{Src}$ are eliminated.

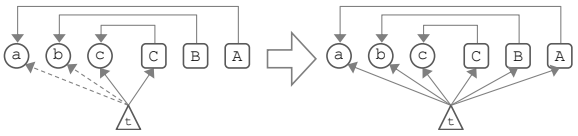


Figure 16: $4 \Rightarrow 0$ und $5 \Rightarrow 0$

Transition: $I_{Src} \wedge II_{Src}$ becomes II_{Src} ($5 \Rightarrow 4$). Figure 17 depicts the reuse pair (B, b) and source requirement a, which has not been reused. Rule TR_2 applies for (C, c): The missing T-link t $\rightarrow_T$ B is set via the RT-linking. Inconsistency I_{Src} for (B, b) is caused by t $\rightarrow_T$ a. Therefore, I_{Src} is not eliminated by RT-linking and TR_3 applies: (C, c) remains II_{Src}.

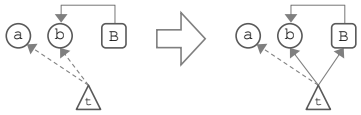


Figure 17: $5 \Rightarrow 4$

Transition: $I_{Tgt} \wedge II_{Src}$ remains ($6 \Rightarrow 6$). Figure 18 shows reuse pair (B, b) and the source requirement a, which was not reused. The T-link t $\rightarrow_T$ B causes (B, b) to be inconsistent I_{Tgt}. Rule TR_4 forces I_{Tgt} to remain. Additionally, TR_3 applies: (C, c) remains II_{Src} because of t $\rightarrow_T$ a.

Transition: $II_{Src} \wedge II_{Tgt}$ stays ($12 \Rightarrow 12$). Figure 18 also shows the reuse pair (C, c). Again, the T-link t $\rightarrow_T$ a causes (C, c) to be unresolvably inconsistent II_{Src} and TR_3 applies. Additionally, TR_4 applies and IITgt remains.

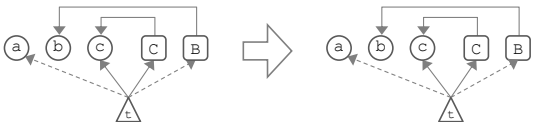


Figure 18: $6 \Rightarrow 6$ und $12 \Rightarrow 12$

C. System B from vehicle series 3 to vehicle series 1

The goal of the first field study was to show that RT-linking is at least as effective as the current manual procedure. To support a full and detailed analysis of the results, the field study was performed on a small-sized system A. The second field study serves another purpose: confirmation.

Confirmation. The second field study was performed on a bigger system B, which had five times more requirements and test cases than system A. The field study was conducted using the same preparations as the first field study. All inconsistency rules were confirmed. No new rules appeared, but more inconsistency transitions did. All new transitions can be described by the four basic inconsistency rules, TR_1, TR_2, TR_3 and TR_4.

D. RT-linking: Conclusion

In Section V, we proposed the basic layer of our 3-layered method to automatically link test cases with reusing requirements. RT-linking uses the assumption:

IF	a target requirement reuses a source requirement
AND IF	a test case verifies a source requirement
THEN	the test case also verifies the target requirement.

Extension of the SOTA. We gave a short introduction into research into requirements traceability in Section III. There we summarized different traceability methods (EBT, RBT, VBT, ...), which automatically create links between requirements and other artefacts. The proposed RT-linking extends the research field via a new method: Reuse-based Test Traceability (RTT).

Supporting industrial practice. RT-linking did not just arise from observing industry practice. We also evaluated the effectiveness of RT-linking under real circumstances with a DOORS extension plug-in. We conducted a field study to compare the overall linking situation of a set of specification documents after manual linking and automatic RT-linking. Thanks to its promising results in terms of accuracy, this plug-in has been transferred to industrial practice. In addition to its accuracy, the complete linking and link analysis of a SRS_{Tgt} now takes a few minutes instead of days or even weeks of manual link creation and maintenance. However, it is clear that some of the T-links have to be reviewed manually. Those T-links which can be established without review define the business case for RT-linking. Several pilot projects showed that these vary between 20% (systems of new vehicle series with many new/changed requirements) and 90% (systems of facelifts with little/no changes). Overall we can say that specification documents of future vehicle series projects will likely be RT-linked.

Outlook: Extension of the RT-linking. The main part of this paper focused on the primary contribution of this work: RT-linking. We will now provide an overview of extensions to RT-linking, as proposed in [1].

VII. Outlook: Test Concept-driven Filtering

RT-linking is the first method layer. It solves the RT-problem. However, real world testing necessitates further considerations. Systematic test planning satisfies both ISO 26262 and testing efficiency. Therefore, we introduce an augmentation to the RT-problem's test case.

Augmentation to the RT-problem. *Test goals* (What purpose?), *test levels* (When?) and *test platforms* (Where?) of the Test Concept [Section II-D] are used to determine the testing expenses. These test planning dimensions describe a three dimensional *testing expenses cube* for each vehicle function. The configuration of each cube defines the testing expenses for a vehicle function and thus also for their refining requirements. Figure 19 depicts the RT-problem, which is augmented with classifying test case properties. Each test case aims to meet specific test goals, is executed during specific test levels and is run on specific test platforms. These classifying test case properties are identical to the test planning dimensions of the *testing expenses cube*. Figure 19 depicts the concept behind the Test Concept-driven Filtering. Test case t is linked with target requirement r_{tgt} if the classifying properties of t fit the configuration of the cube. In this context, new RT-inconsistencies arise, for example: (1) If a test case meets more test goals than demanded by the cube for the vehicle function then too many test goals are met. Thus, the testing is not minimally efficient. (2) If the test cube demands more test goals then all T-linked test cases put together, then too few test goals will be met. As a result, the testing is not complete.

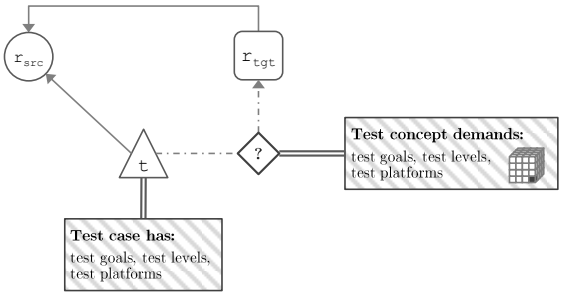


Figure 19: Test planning dimensions of the RT-problem

Extension of RT-linking. Test Concept-driven Filtering extends RT-linking as follows:

IF	a target requirement reuses a source requirement
AND IF	a test case verifies a source requirement
THEN	the test case also verifies the target requirement according to the Test Concept.

This second method layer has been proposed in [34]. Further, the filtering technique has been implemented in DOORS DXL. It is currently being evaluated in an industrial environment.

VIII. Outlook: Case-based Filtering

In the second method layer, we only augmented the test case of the RT-problem in order to connect it with the Test Concept. The third layer eliminates this unused potential by *fully* augmenting the RT-problem. In this way, we facilitate similarity between RT-problems.

Full augmentation of the RT-problem. Figure 19 depicts the test case augmentation with classifying test case properties. Figure 20 depicts the additional augmentation with classifying requirements properties. These requirements and test properties can be defined freely, e.g., ASIL ranking, interfaces, requirements maturity, OEM or supplier status, test case derivation method, ownership, etc. By virtue of the R-link between requirements, property changes can be detected from source to target in relation to the test properties. As the following example illustrates, this presents countless possibilities. We assume that the owner of a source requirement and a T-linked test case is *Thomas*. Further, we assume that the owner of the target requirement has changed to *Jonathan*. Case-based Filtering allows us to detect such situations. The detection mechanism adapts the retrieval techniques of Case-based reasoning as follows: An RT-case is structurally identical to an RT-problem, except for an additional RT-decision and review note. More precisely, Figure 20 does not depict an RT-problem but an RT-case. We use a similarity function to assess the similarity of the classifying properties between an RT-problem and an RT-case. Thus, we not only detect similar RT-problems for a given RT-case, but also adapt the RT-decision and review note to solve the RT-problem.

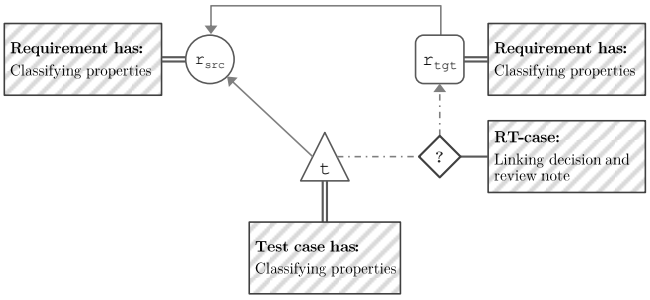


Figure 20: Classifying properties of the RT-problem

Extension of RT-linking. Case-based Filtering extends RT-linking as follows:

IF	a target requirement reuses a source requirement
AND IF	a test case verifies a source requirement
THEN	the test case also verifies the target requirement according to the RT-Case-basis.

A prototype of the third layer has been implemented in DOORS DXL. An initial presentation sparked the interest of requirements and test engineers.

IX. Conclusions and Future Work

In this paper, we proposed RT-linking, the main technique of our 3-layered method, to automatically solve the RT-problem. RT-problems occur after requirements reuse (R) but before test reuse (T). The main goal is to reuse test cases on the basis of requirements reuse.

Comprehensive requirements reuse. We proposed a new trace link type: The Reuse- or R-link, which connects reusing target requirements with reused source requirements. These R-links facilitate comprehensive reuse traceability as demanded by ISO 26262 [3, p.20]. By virtue of R-links we are able to systematically distinguish between new or discarded requirements and requirements which have been reused with or without modification.

Comprehensive test specification. Test cases are trace linked with requirements by Test- or T-links to facilitate test traceability according to ISO 26262 [2, p.25]. These T-links are influenced by the requirements reuse; reuse of requirements implies reuse of test cases. In this paper, we proposed transitive RT-linking to set T-links automatically from test cases to reusing target requirements on the basis of previously T-linked source requirements. Thanks to the promising results from the field studies, RT-linking has been transferred to industrial practice.

Comprehensive test planning and decisions. Although this paper focused on the first method layer, we gave a short overview of the second and third method layers. Both layers provide additional filter techniques to improve the RT-linking technique. The second layer arranges RT-linking according to test planning, while the third layer defines and uses similarity between RT-problems to reuse link decisions.

Rotate and tilt the RT-problem. Outside of industrial application, an interesting idea has arisen. We used requirements, test cases and specific link types to describe the RT-problem. However, it could be much more general than this if we used abstract artefacts and abstract link types instead. V-Models are usually characterized by many different trace linked artefacts. As well as requirements and test cases, systems, components, safety cases, architecture, feature models, functional models and code also exist, among others. Even the SOTA often relies on holistic traceability models [Section III-A]. Future work could focus on the interesting question of whether a general RT-problem is able to support a traceability model by successively rotating or tilting it through the V-Model, each followed by a general RT-linking step.

X. Acknowledgements

We thank our DCAITI colleagues Quang Minh Tran, Jonas Winkler and Martin Beckmann for discussions and proof reading. Furthermore, we thank our Daimler colleagues for positively impacting our concepts and implementations with respect to relevance, feasibility and usability.

References

[1] T. Noack, "Automatic Linking of Test Cases and Requirements," in Proceedings of the 5th International Conference in System Testing and Validation Lifecycle (VALID), Venice, Italy, 2013, pp. 45–48.

[2] Road Vehicles - Functional Safety, Part 8: Supporting processes (ISO 26262-8:2011), International Organization for Standardization, 2011.

[3] Road Vehicles - Functional Safety, Part 6: Product Development: Software Level (ISO 26262-6:2011), International Organization for Standardization, 2011.

[4] A. Spillner and T. Linz, Basiswissen Softwaretest, 4th ed. Heidelberg: dpunkt.verlag, 2005.

[5] Software Engineering - Product Quality, Part 1: Quality Model (ISO 9126-1:2001), European Committee for Standardization and International Organization for Standardization, 2005.

[6] R. Watkins and M. Neal, "Why and How of Requirements Tracing," IEEE Software, vol. 11, no. 4, 1994, pp. 104–106.

[7] J. Cleland-Huang, O. Gotel, and A. Zisman, Software and Systems Traceability, 1st ed. Springer, 2012.

[8] F. A. C. Pinheiro, "Requirements Traceability," in Perspectives on Software Requirements, 1st ed., J. C. S. do Prado Leite and J. H. Doorn, Eds. Kluwer Academic Publishers, 2004, ch. 5, pp. 91–113.

[9] K. Pohl and P. Haumer, "HYDRA: A Hypertext Model for Structuring Informal Requirements Representations," in Proceedings of the 2nd International Workshop on Requirements Engineering (REFSQ), K. Pohl and P. Peters, Eds. Jyväskylä, Finnland: Verlag der Augustinus-Buchhandlung, 1995, pp. 118–134.

[10] E. Horowitz and R. C. Williamson, "SODOS : A Software Documentation Support Environment - Its Definition," IEEE Transactions on Software Engineering, vol. 12, no. 8, 1986, pp. 849–859.

[11] J. Conklin and M. L. Begeman, "glBIS : A Hypertext Tool for Team Design Deliberation," in Proceedings of the ACM Conference on Hypertext. Chapel Hill, NC, USA: ACM Press, 1987, pp. 247–251.

[12] B. Ramesh and V. Dhar, "Supporting Systems Development by Capturing Deliberations During Requirements Engineering," IEEE Transactions on Software Engineering, vol. 18, no. 6, 1992, pp. 498–510.

[13] H. Kaindl, "The Missing Link in Requirements Engineering," ACM SIGSOFT Software Engineering Notes, vol. 18, no. 2, 1993, pp. 30–39.

[14] F. A. C. Pinheiro and J. A. Goguen, "An Object-Oriented Tool for Tracing Requirements," in Proceedings of the 2nd International Conference on Requirements Engineering. Colorado Springs, CO, USA: IEEE Computer Society Press, 1996, pp. 52–64.

[15] T. Tsumaki and Y. Morisawa, "A Framework of Requirements Tracing using UML," in Proceedings of 7th Asia-Pacific Software Engineering Conference (APSEC). Singapur, Singapur: IEEE Computer Society Press, 2000, pp. 206–213.

[16] P. Letelier, "A Framework for Requirements Traceability in UML-Based Projects," in Proceedings of 1st International Workshop on Traceability in Emerging Forms of Software Engineering, Edinburgh, Schottland, 2002, pp. 30–41.

[17] M. Narmanli, "A Business Rule Approach to Requirements Traceability," Masterarbeit, Middle East Technical University, 2010.

[18] B. Turban, M. Kucera, A. Tsakpinis, and C. Wolff, "Bridging the Requirements to Design Traceability Gap," Intelligent Technical Systems, Lecture Notes in Electrical Engineering, vol. 38, 2009, pp. 275–288.

[19] S. Ibrahim, M. Munro, and A. Deraman, "Implementing a Document-Based Requirements Traceability: A Case Study," in Proceedings of International Conference on Software Engineering, P. Kokol, Ed. Innsbruck, Österreich: ACTA Press, 2005, pp. 124–131.

[20] H. U. Asuncion, F. Francois, and R. N. Taylor, "An End-To-End Industrial Software Traceability Tool," in Proceedings of the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering. Dubrovnik, Kroatien: ACM, 2007, pp. 115–124.

[21] A. Azmi and S. Ibrahim, "Implementing Test Management Traceability Model to Support Test Documents," International Journal of Digital Information and Wireless Communications (IJDIWC), vol. 1, no. 1, 2011, pp. 109–125.

[22] J. MacMillan and J. R. Vosburgh, "Software Quality Indicators," Scientific System Inc., Cambridge MA., Tech. Rep., 1986.

[23] DOORS (Dynamic Object Oriented Requirements System) , IBM, 2013.

[24] S. Rochimah, W. M. N. Wan Kadir, and A. H. Abdullah, "An Evaluation of Traceability Approaches to Support Software Evolution," in International Conference on Software Engineering Advances (ICSEA 2007). Cap Esterel, Frankreich: IEEE Computer Society, Aug. 2007, pp. 19–38.

[25] R. Torkar, T. Gorschek, R. Feldt, M. Svahnberg, U. A. Raja, and K. Kamran, "Requirements Traceability: A Systematic Review and Industry Case Study," International Journal of Software Engineering and Knowledge Engineering, vol. 22, no. 3, 2012, pp. 385–433.

[26] J. Cleland-Huang, C. K. Chang, and M. Christensen, "Event-Based Traceability for Managing Evolutionary Change," IEEE Transactions on Software Engineering, vol. 29, no. 9, 2003, pp. 796–810.

[27] G. Spanoudakis, A. Zisman, E. Pérez-Minana, and P. Krause, "Rule-Based Generation of Requirements Traceability Relations," Journal of Systems and Software, vol. 72, no. 2, Jul. 2004, pp. 105–127.

[28] M. Riebisch, "Supporting Evolutionary Development by Feature Models and Traceability Links," in 11th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems (ECBS). Brno, Tschechien: IEEE Computer Society Press, 2004, pp. 370–377.

[29] M. Heindl and S. Biffl, "A Case Study on Value-Based Requirements Tracing," in Proceedings of the 10th European Software Engineering Conference held jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering. Lissabon, Portugal: ACM Press, 2005, pp. 60–69.

[30] A. Egyed and P. Grünbacher, "Supporting Software Understanding with Automated Requirements Traceability," International Journal of Software Engineering and Knowledge Engineering, vol. 15, no. 5, 2005, pp. 783–810.

[31] J. Cleland-Huang, R. Settimi, O. BenKhadra, E. Berezhanskaya, and S. Christina, "Goal-Centric Traceability for Managing Non-Functional Requirements," in Proceedings of the 27th International Conference on Software Engineering (ICSE). St. Louis, MO, USA: ACM Press, 2005, pp. 362–371.

[32] R. Oliveto, M. Gethers, D. Poshyvanyk, and A. De Lucia, "On the Equivalence of Information Retrieval Methods for Automated Traceability Link Recovery," in 18th IEEE International Conference on Program Comprehension (ICPC). Minho, Portugal: IEEE Computer Society Press, Jun. 2010, pp. 68–71.

[33] A. Cmyrev, R. Nörenberg, D. Hopp, and R. Reissing, "Consistency Checking of Feature Mapping between Requirements and Test Artefacts," Concurrent Engineering Approaches for Sustainable Product Development in a Multi-Disciplinary Environment, vol. 1, no. 1, 2013, pp. 121–132.

[34] T. Noack, "Automatische Verlinkung von Testfällen und Anforderungen: Testplangesteuerte Filterung," Softwaretechnik-Trends, vol. 33, no. 4, 2013, pp. 5–6.

[35] SimMetrics (Open Source library with numerous algorithms to calculate textual similarity between two texts), Source Forge, 2014.