Bidirectional Requirements Traceability

By Linda Westfall

the WestfallTeam
Partnering for Software Excellence

www.westfallteam.com

Traceability is one of the essential activities of good requirements management. Traceability is used to ensure that the right products are being built at each phase of the software development life cycle, to trace the progress of that development and to reduce the effort required to determine the impacts of requested changes. This article explores:

- What is traceability?
- Why is traceability a good practice?
- How is traceability performed?

What is Traceability?

The IEEE Standard Glossary of Software Engineering Terminology defines traceability as "the degree to which a relationship can be established between two or more products of the development process, especially products having a predecessor-successor or master-subordinate relationship to one another." [IEEE-610]

Traceability is used to track the relationship between each unique product-level requirement and its source. For example, a product requirement might trace from a business need, a user request, a business rule, an external interface specification, an industry standard or regulation, or to some other source.

Traceability is also used to track the relationship between each unique product-level requirement and the work products to which that requirement is allocated. For example, a single product requirement might trace to one or more architectural elements, detail design elements, objects/classes, code units, tests, user documentation topics, and/or even to people or manual processes that implements that requirement.

Good traceability practices allow for bidirectional traceability, meaning that the traceability chains can be traced in both the forwards and backwards directions as illustrated in Figure 1.

Forward traceability looks at:

- Tracing the requirements sources to their resulting product requirement(s) to ensure the completeness of the product requirement specification.
- Tracing each unique product requirement forward into the design that implements that requirement, the code that implements that design and the tests that validate that requirement and so on. The objective is to ensure that each requirement is implemented in the product and that each requirement is thoroughly tested.

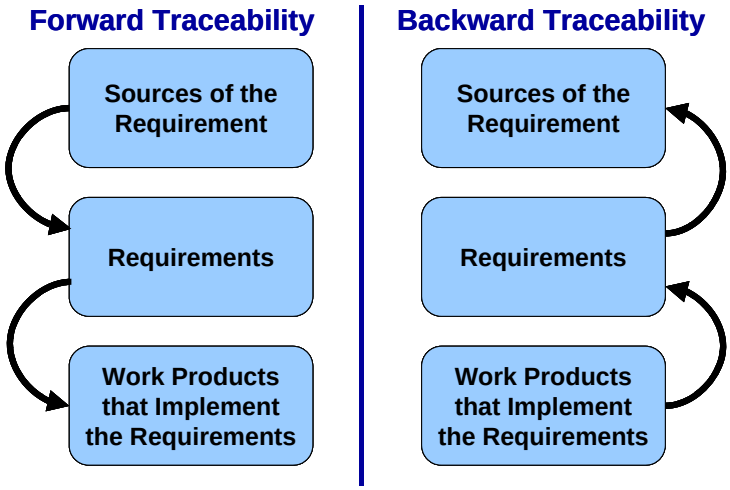

Figure 1: Bidirectional (Forward & Backward) Traceability

Backwards traceability looks at:

- Tracing each unique work product (e.g., design element, object/class, code unit, test) back to its associated requirement. Backward traceability can verify that the requirements have been kept current with the design, code, and tests.
- Tracing each requirement back to its source(s).

Why is Traceability a Good Practice?

The Software Engineering Institute (SEI) Capability Maturity Model Integration® (CMMI®) states that the purpose of the process area in "Requirements Management is to manage the requirements of the project's product and product components and to identify inconsistencies between those requirements and the project's plans and work products." [SEI-00]. One of the specific practices under the Requirements Management process area is to "Maintain Bidirectional Traceability of Requirements." [SEI-00] What is the benefit of putting in the effort to maintain bidirectional traceability?

Forward traceability ensures proper direction of the evolving product (that we are building the right product) and indicates the completeness of the subsequent implementation. For example, if a business rule can't be traced forward to one or more product requirements then the product requirements specification is incomplete and the resulting product may not meet the needs of the business. If a product requirement cannot be traced forward to its associated architectural design elements, then the architectural design is not complete and so on.

Backwards traceability helps ensure that the evolving product remains on the correct track with regards to the original and/or evolving requirements (that we are building the product right). The objective is to ensure that we are not expanding the scope of the project by adding design elements, code, tests or other work products that are not called out in the requirements (i.e., "gold plating"). If there is a change needed in the implementation or if the developers come up with a creative, new technical solution, that change or solution should be traced backwards to the requirements and the business needs to ensure that it is within the scope of the desired product. For example, if there is a work product element that doesn't trace backwards to the product requirements one of two things is true. The first possibility is that there is a missing requirement because the work product element really is needed. In this case, traceability has helped identify the missing requirement and can also be used to evaluate the impacts of adding that requirement to project plans and other work products (forward traceability again). The second possibility is that there is "gold plating" going on – something has been added that should not be part of the product. Gold plating is a high risk activity because project plans have not allocated time or resources to the work and the existence of that part of the product may not be well communicated to other project personnel (e.g., tester doesn't test it, it's not included in user documentation).

Forward and backward traceability can also be used to assess the impact of requirement changes. If there are changes in the business environment (for example, a business objective, stakeholder requirement or standard has changed), then if good forward traceability has been maintained, that change can be traced forward to the associated requirements and all of the work products that are impacted by that change. This greatly reduces the amount of effort required to do a thorough job of impact analysis. It also reduces the risk that one of the effected work products is forgotten, resulting in an incomplete implementation of the change (i.e., a defect). If, on the other hand, a requirement change request is received that traces back to a government regulation and that regulation has not changed, it may result in the rejection of the requested change.

Another benefit of backward traceability comes when a defect is identified in one of the work products. For example, if a piece of code has a defect, the traceability matrix can be used to help determine the root cause of that defect. Is it just a code defect or does it trace back to a defect in the design or requirements? As illustrated in Figure 2, not only does unit ABC have the defect, but the traceability matrix can be used to help determine that defects also exist in design element A and requirements R1302.

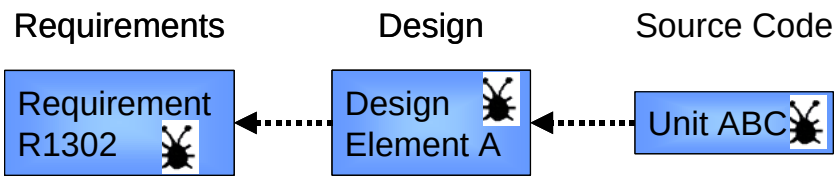

Figure 2: Backwards Traceability and Impact Analysis

If design or requirement's defect are discovered during this analysis, what other work products might be impacted by the defect? Forward traceability can be used to ensure a complete analysis. Figure 3 continues the impact analysis example by illustrating the use of this forward traceability. In this example, the defective requirement traced forward to two additional design elements Y, which had the defect and X, which did not. Design elements A and Y trace forward into additional source code units DEF and Y02, which also contained the defect (and unit Y01 which did not have the defect)

The requirements also traced forward to:

- Two training models, one of which had the defect
- Two system test cases, one of which had the defect
- One User document with the defect

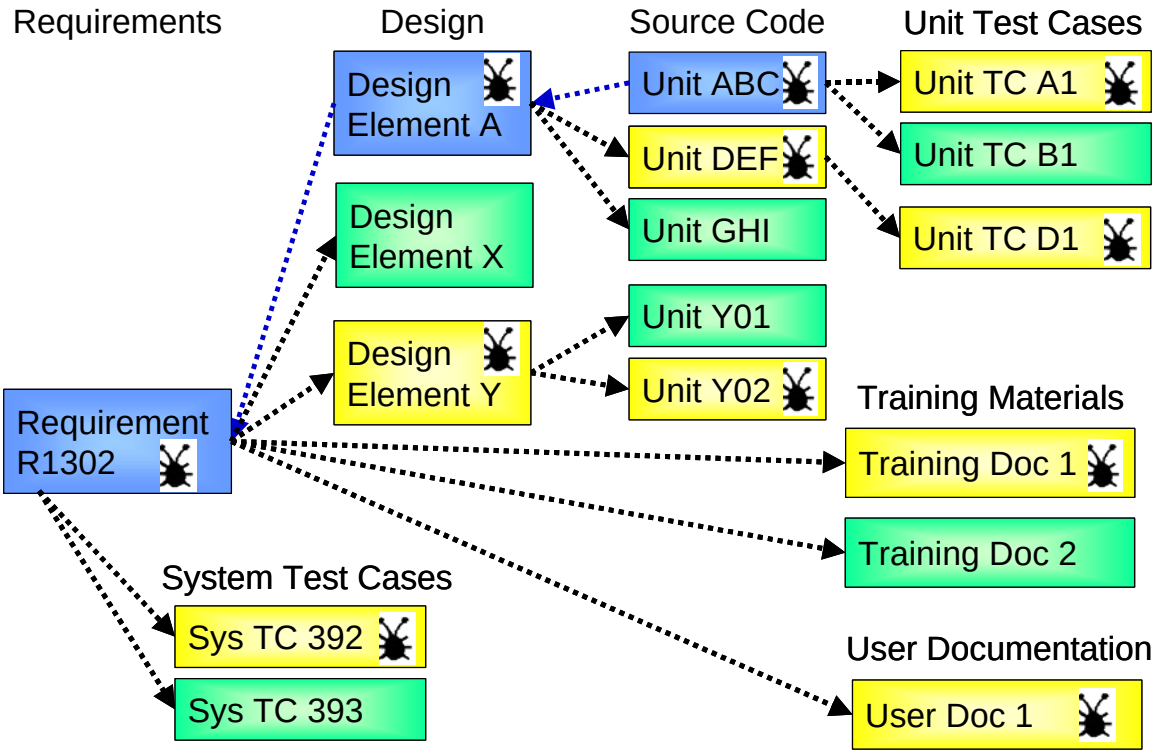

Figure 3: Forwards Traceability and Impact Analysis

Benefits of bi-directional requirements traceability include the ability to:

- Analyze the impact of a change
  - All work products affected by a changed requirement
  - All requirements affected by a change or defect in a work product
- Assess current status of the requirements and the project
  - Identify missing requirements
  - Identify gold plating

How is Traceability Performed?

The classic way to perform traceability is by constructing a traceability matrix. As illustrated in Table 1, a traceability matrix summarizes in matrix form the traceability from original identified stakeholder needs to their associated product requirements and then on to other work product elements. In order to construct a traceability matrix, each requirement, each requirements source and each work product element must have a unique identifier that can be used as a reference in the matrix. The requirement matrix has the advantage of being a single repository for documenting both forwards and backwards traceability across all of the work products.

Requirement Source	Product Requirements	HLD Section #	LLD Section #	Code Unit	UTS Case #	STS Case #	User Manual
Business Rule #1	R00120 Credit Card Types	4.1 Parse Mag Strip	4.1.1 Read Card Type	Read_Card_Type.c Read_Card_Type.h	UT 4.1.032 UT 4.1.033 UT 4.1.038 UT 4.1.043	ST 120.020 ST 120.021 ST 120.022	Section 12
			4.1.2 Verify Card Type	Ver_Card_Type.c Ver_Card_Type.h Ver_Card_Types.dat	UT 4.2.012 UT 4.2.013 UT 4.2.016 UT 4.2.031 UT 4.2.045	ST 120.035 ST 120.036 ST 120.037 ST 120.037	Section 12
Use Case #132 step 6	R00230 Read Gas Flow	7.2.2 Gas Flow Meter Interface	7.2.2 Read Gas Flow Indicator	Read_Gas_Flow.c	UT 7.2.043 UT 7.2.044	ST 230.002 ST 230.003	Section 21.1.2
	R00231 Calculate Gas Price	7.3 Calculate Gas price	7.3 Calculate Gas price	Cal_Gas_Price.c	UT 7.3.005 UT 7.3.006 UT 7.3.007	ST 231.001 ST 231.002 ST 231.003	Section 21.1.3

Table 1: Example of a Traceability Matrix

Modern requirements management tools include traceability mechanisms as part of their functionality.

A third mechanism for implementing traceability is trace tagging. Again, each requirement, each requirement source and each work product element must have a unique identifier. In trace tagging however, those identifiers are used as tags in the subsequent work products to identify backwards tracing to the predecessor document. As illustrated in Figure 2 for example, the Software Design Specification (SDS) includes tags that identify the requirements implemented by each uniquely identified design element and the Unit Test Specification (UTS) includes trace tags that trace back to the design elements that each test case verifies. This tagging propagates through the work product set with source code units that include trace tags back to design elements, integration test cases with tags back to architecture elements and system test cases with trace tags back to requirements as appropriate depending on the hierarchy of work products used on the product. Trace tags have the advantage of being part of the work products so a separate matrix isn't maintained. However, while backwards tracing is easy with trace tags, forward tracing is very difficult using this mechanism.

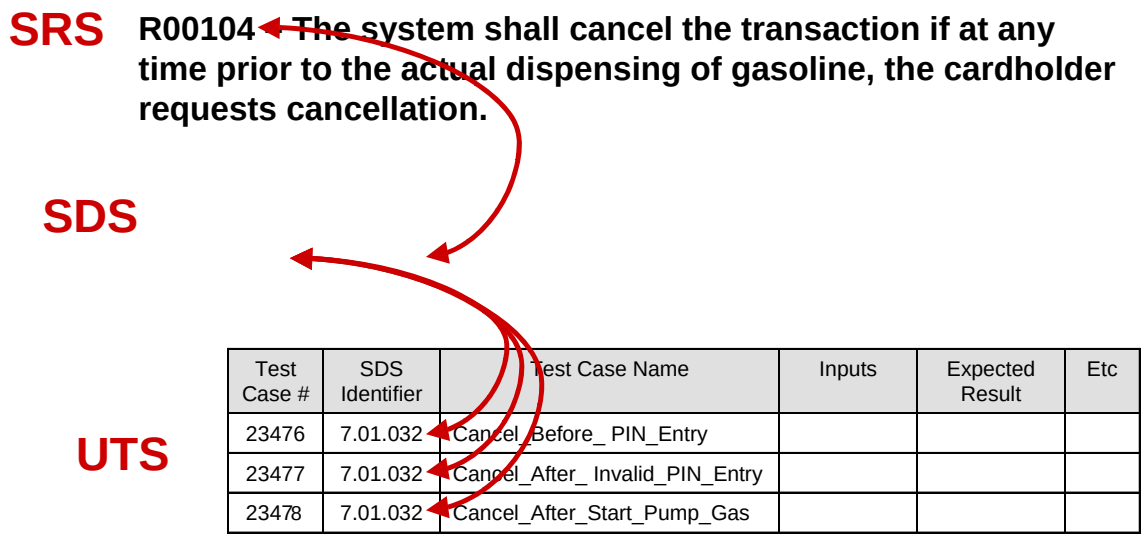

Figure 4: Example of Trace Tagging

All traceability implementation techniques require the commitment of a cross-functional team of participants to create and maintain the linkages between the requirements, their source and their allocation to subsequent work products. The requirements analyst must initiate requirements traceability and document the original tracing of the product requirements to their source. As system and software architects create the high-level design, those practitioners add their information to the traceability documentation. Developers doing low-level design, code and unit testing add additional traceability information for the elements they create, as do the integration, system and alpha, beta and acceptance testers. For small projects, some of these roles may not exist or may be done by the sample practitioner, which limits the number of different people working with the traceability information. For larger projects, where traceability information comes from many different practitioners, it may be necessary to have someone who coordinates, documents and ensures periodic audits of the traceability information from all its various sources to achieve completeness and consistency.

References

IEEE-610 IEEE Standards Software Engineering, *IEEE Standard Glossary of Software Engineering Terminology, IEEE Std. 610-1990*, The Institute of Electrical and Electronics Engineers, 1999, ISBN 0-7381-1559-2.

SEI-00 *CMMI[SM] for Systems Engineering/Software Engineering, Version 1.02 (CMMI-SW/SW, V 1.02); CMMI Staged Representation, CMU/SEI-2000-TR-018, ESC-TR-2000-018; Continuous Representation, CMU/SEI-2000-TR-019, ESC-TR-2000-019;* Product Development Team; Software Engineering Institute; November 2000.