

Vertical and Horizontal Requirements Relationships

By Theresa Hunt



www.westfallteam.com

This article explores a common question raised about Capability Maturity Model – Integration (CMMI[®]) expectations regarding the management of vertical and horizontal relationships within the context of requirements traceability. It seems that the CMMI[®] really does not define vertical and horizontal traceability explicitly. In order to document/prove/provide evidence of vertical and/or horizontal requirements traceability, we first need an understanding of exactly what vertical and horizontal traceability are in the context of software systems. In hopes of finding an acceptable answer, it is necessary for this paper to:

- First examine the statement in question and how it has changed between versions 1.1 and 1.2 of the CMMI[®]
- Second, research how the CMMI[®] defines traceability and how the definitions have changed between the most recent versions
- Third, attempt to understand how the CMMI[®] intends for us to interpret the meaning
- Finally put all of this into perspective with an example

The Statement in Question

In the requirements management process area under Specific Process (SP) area 1.4 “Maintain bidirectional traceability among the requirements and work products” we find the following statement:

Old statement: (CMMI-SE/SW/IPPD/SS, V1.1)

“The traceability should cover both the horizontal and vertical relationships, such as across interfaces”

New statement (CMMI-DEV, V1.2):

“The traceability can cover horizontal relationships, such as across interfaces, as well as vertical relationships”

The new statement makes clear that the example “such as across interfaces” applies to the horizontal relationships.

How the CMMI[®] Defines Traceability

Table 1 – Terminology Changes, contains excerpts from versions 1.1 and 1.2 of the CMMI[®] glossary. One of the three definitions, “bidirectional traceability”, is new, and the other two “requirements traceability” and “traceability” have been modified as shown.

Bidirectional traceability	NEW V1.2 TERM
	An association among two or more logical entities that is discernable in either direction (i.e., to and from an entity). (See also “requirements traceability” and “traceability.”)

Requirements traceability	<p>OLD V1.1 DEFINITION: The evidence of an association between a requirement and its source requirement, its implementation, and its verification.</p> <p>NEW V1.2 DEFINITION: A discernable association between requirements and related requirements, implementations, and verifications. (See also “bidirectional traceability” and “traceability.”)”</p>
Traceability	<p>OLD V1.1 DEFINITION: (See “requirements traceability.”)</p> <p>NEW V1.2 DEFINITION: A discernable association among two or more logical entities such as requirements, system elements, verifications, or tasks. (See also “bidirectional traceability” and “requirements traceability.”)</p>

Table 1 – Terminology Changes

Definition Interpretation

In searching other SEI publications for subject matter related to requirements traceability, I found the following statement in an SEI technology description of requirements tracing under the category of “strengths” for one of the approaches: “*fine-grained forward, backward, horizontal, and vertical RT*”. This statement includes four very distinct terms, so the assumption is that they are to be interpreted as having four very distinct implications.

For simplicity lets assume that “forward and backward” traceability means the same thing as *bidirectional* traceability (see Figure 1 - bidirectional traceability). Common practice is to document bidirectional traceability in “forward” and “backward” traceability matrices. Forward traceability shows that each originating requirement traces forward into the design, code, test, and other work product(s) that implement that requirement. Backward traceability shows that each work product traces backwards to its associated predecessor work product(s) and requirement(s) of origin.

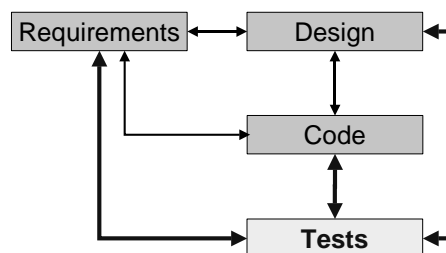


Figure 1 – Bidirectional Traceability

Bidirectional traceability is very useful in ensuring that we are not expanding the scope of the project by adding design elements, code or tests that are not called out in the requirements and that the requirements have been kept current with the design, code, and tests. If we end up with a piece of functionality in the software that cannot be traced back to a requirement, we have one of two possible situations: one – there is a missing requirement, or two – there is gold plating (superfluous functionality not intended or budgeted for). For a more detailed discussion of bidirectional traceability see Linda Westfall’s article “Bidirectional Requirements Traceability” [Westfall-06]. To further explore industry standard definitions we could consider the IEEE-610 glossary, which says that traceability is

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.

From that definition we might ask the following questions: Does the predecessor-successor relationship mean horizontal (forward and backward (bi-directional) traceability)? Does the master-subordinate relationship mean vertical traceability (up and down)? The answer I think depends on your view of “direction”; in other words, these terms are clearly open to interpretation depending on a number of variables, for example, whether the development activities were performed using plan-driven or agile methodologies, etc.

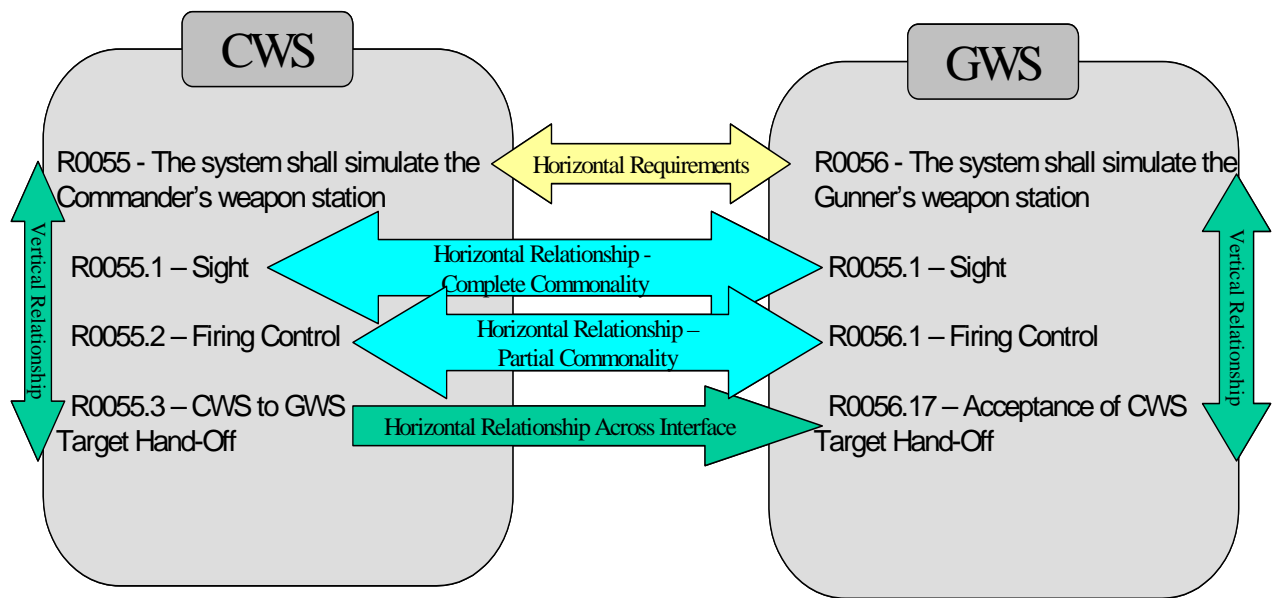
What we can say with certainty is that the two-dimensional traceability matrix is very useful in depicting high-level bidirectional traceability. What we can't say, just by looking at the matrix, is the extent or dimension of that traceability (“*the degree to which*” in the IEEE-610 definition). What if a particular capability only partially satisfies a requirement? Or what if a particular capability satisfies more than one requirement? In other words, the matrix just shows that one or more requirements are mapped to one or more entities – not *which parts* of which requirements are addressed in *which parts* of which entities. Nor does it show similarities or where one entity has the potential to affect another. This is where I believe the heart of the issue with vertical and horizontal relationships lie and where I believe we can take some of the obscurity out of the CMMI direction that our traceability should cover vertical and horizontal relationship mapping. We can use vertical and horizontal relationship indicators to indicate in more depth how functionality is distributed among constituent capabilities. Another important use for this information is the design of appropriate test cases that align and distribute individual test procedures with code.

Example

To illustrate how we might reconcile vertical and horizontal relationship traceability with the more commonly utilized bidirectional traceability, it's necessary to try and work them in together using an example that will hopefully put things into perspective. As our example system, we will utilize a simulation device used to train military troops (or if you prefer, a video game). Typically, a high level or basal system requirement such as “The system shall simulate the Commander's Weapon Station” will have an allocation to software (i.e., Commander's Weapon Station (CWS) simulation), which will be specified along with its constituent lower level and derived software requirements (sight, firing control, etc). Subsequently these lower level and derived software requirements will be allocated to the specific software work products that implement them (design, code, tests) in a classical vertical relationship. A corresponding horizontal *requirement* could be one that is similar in position, purpose, form, etc. that is essentially a variant or version of the same type of information, for example, “The system shall simulate the Gunner's weapon station” (i.e., Gunner's Weapon Station (GWS) simulation), which will also be specified along with its levels of constituent lower level and derived software requirements (sight, firing control, etc). During design, the sharing (or level of similarity) of the functions of sight, firing control, etc. might be designated as complete (the same code module is used by both CWS and GWS) or partial (slight customization by either CWS or GWS) so that both the CWS and the GWS have as little redundant work products in their implementation as possible. A horizontal mapping of requirements to implementation work product coverage is indicated. By “coverage” I mean that the mapping should indicate *which parts* of which requirements are mapped to *which parts* of which entities. Similar issues would evidence themselves in existing reusable components in external source libraries that are used by more than one capability in achieving their required functionality. The key is to ensure that all interface points have been identified. The traceability matrix would need a method for depicting these relationships such as the use of flags to indicate an association, the use of nomenclature rules, bipartition mapping of the requirements and associated work products, etc.

There may also be a requirement for the Commander to be able to hand off a target to the Gunner, which creates an interface point between corresponding requirements (a horizontal *relationship*). It is a specific requirement, but how do you architecturally allocate and trace it?

We allocate it to one software entity, but as with the sight and fire control functions we should be able to track that it crosses interfaces to the other entity.



Summary

Clearly, both horizontal and vertical traceability can be bidirectional. As for discerning their “difference” I would proffer that vertical requirements traceability is a “provider, recipient” type of association – similar to a parent, child, or grandchild relationship and that horizontal requirements traceability is a “version, variant, interaction” type of association – similar to a sibling, aunt, uncle, or cousin relationship. Plausible definitions might then be:

Vertical requirements traceability: the depiction of discernable hierarchical associations between requirements and resultant work products.

Horizontal requirements traceability: the depiction of discernable nonhierarchical similarities, shared properties, data, interactions, etc. among requirements and work products.

Establishing and maintaining robust vertical and horizontal requirements traceability enhances our ability to perform effective and complete impact analysis and regression testing of the system when the software changes (e.g., when defects are identified or requirements change). By examining all of the associations that a modified work product has with other work products we are in a better position to find adverse side effects and ensure that the change did not impede compliance with specified requirements. We must also be able to trace propagation of a change to all affected products, customers/users, sites and releases.

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.
- CMMI V1.1 © Capability Maturity Model Integration [Http://www.sei.cmu.edu](http://www.sei.cmu.edu)
- CMMI V1.2 © Capability Maturity Model Integration [Http://www.sei.cmu.edu](http://www.sei.cmu.edu)
- SEI Tech Des http://www.sei.cmu.edu/str/descriptions/reqtracing_body.html
- Westfall-06 Linda Westfall "Bidirectional Requirements Traceability"
http://www.westfallteam.com/Papers/Bidirectional_Requirements_Traceability.pdf
- © Capability Maturity Model Integration, Capability Maturity Model, Capability Maturity Modeling, CMMI, and CMM are registered in the U.S. Patent & Trademark Office.
- SEI are service marks of Carnegie Mellon University.