Revisiting the Fundamentals of Traceability between Requirements and its Realization

Anitha Murugesan, Michael Whalen, Elaheh Ghassabani and Mats Heimdahl

Department of Computer Science and Engineering

University of Minnesota, 200 Union Street, Minneapolis, Minnesota 55455

{anitha, whalen, ghass013,heimdahl}@cs.umn.edu

***Abstract*—When establishing associations, known as *trace links*, between a requirement and the artifacts that lead to its satisfaction, it is essential to know what the links mean. While research into this type of traceability, that we call *Requirements Satisfaction Traceability*, has been an active research area for some time, none of the literature that we have found discusses the fact that there are often multiple possible satisfaction arguments that each use a different set of implementation elements. The distinction between establishing a single satisfaction argument between a requirement and its implementation (tracing *one* way the requirement is implemented) vs. all satisfaction arguments, and the possible ramifications for how the trace links can be used in analysis, has not been well studied. We examine how this distinction changes the way traceability is perceived, established, maintained, and used. We introduce and discuss the notion of "complete" traceability, which considers *all* trace links between the requirements and the artifacts that work to satisfy the requirements, and contrast it with the partial traceability common in practice. The intent of this RE@Next! paper is to highlight the aspect of completeness in traceability and discuss the ramifications of incomplete (or partial) traceability.**

I. Introduction

Requirements traceability – "the ability to describe and follow the life of a requirement, in both a forwards and backwards direction (i.e., from its origins, through its development and specification, to its subsequent deployment and use, and through all periods of on-going refinement and iteration in any of these phases)." [2] – has been a topic of great interest in research and practice for several decades. Intuitively, it concerns establishing relationships, called *trace links*, between the requirements and one or more artifacts of the system. Among the several different development artifacts and the relationships that can established from/to the requirements, being able to establish trace links from requirements to artifacts that realize or *satisfy* those requirements – particularly to entities within those artifacts, that we will call *target artifacts* in the rest of the paper – has been enormously useful in practice. For instance, it helps analyze the impact of changes in one artifact

on the other, assess the quality of the system, aid in creating assurance arguments for the system, etc. In this paper, we focus our attention to this subset of requirement traceability, that we call *Requirements Satisfaction Traceability.*

Instead of just recording the trace links from each requirement to the target artifacts, *Satisfaction Arguments* [5] offer a semantically rich way to establish them. Originally proposed by Zave and Jackson, a satisfaction argument demonstrates how the behaviors of the system and its environment together satisfy the requirements. From a traceability perspective, these arguments help establish trace links (the *satisfied by* relationship) between the requirements and those parts of the system and environment (the target artifacts) that were necessary to satisfy the requirements; We call those target artifacts the *set of support* for that requirement. Mathematically, if we think of the argument as a proof in which the requirement is the claim, then the set of support are the axioms (or clauses) that were necessary to prove the claim and the trace links are means to associate the claim to those clauses. Such proofs are not, in general, unique, and often there are multiple sets of clauses that could be used to construct the proof. Neither Zave and Jackson nor (in our examination) the existing traceability literature discusses multiple alternative satisfaction arguments or sets of trace links for one system design.

While it is standard practice to refer to describe a satisfaction argument as capturing the relationship between artifacts, in our opinion, at least when discussing requirements satisfaction traceability, it is beneficial to distinguish between trace links to "a" set of support vs. "the" sets of support. In so doing, it encourages a substantial change in understanding what traceability *means* and how traceability be used. For instance, in a safety critical system, one could intentionally add multiple ways to satisfy a requirement for fault tolerance or one could unintentionally add system functionality that leads to multiple ways of satisfying a requirement. Although for proof of satisfaction, one set of support is sufficient, it is insufficient for other interesting analyses that can be performed using trace information. For example, when there is a change in an implementation element required by the argument, impact analysis may be overly conservative; alternate paths to satisfy the requirement may be unaffected. In addition, when comparing multiple approaches toward constructing satisfaction arguments (e.g., manual approaches vs. automated or semi-automated approaches), it is possible that the approaches are constructing different, but equally valid arguments.

Establishing trace links to all sets of support, that we call *complete* traceability, provides us insight about the elements of the set of support - ones that are necessary, the ones that are optional and the ones that do not contribute to satisfying a requirement. We categorize the elements as *Must*, *May* and *Irrelevant* support elements for each requirement. This categorization of support elements is useful in several ways, such as (a) to precisely perform impact analysis, (b) to determine whether a change to a system element affects the ways requirements are satisfied, *Mike: Is (b) different than (a)?* (c) to identify which parts of the system are not necessary to satisfy any requirement of the system, so called "Gold Plating", (d) to locate parts of the system that are most critical

in terms of satisfying most requirements, (e) to identify system assumptions that are used by most requirements, and so on. In fact, establishing complete traceability also may help to identify unintended ways the requirement was satisfied by the system and its environment.

In this paper, we examine how this notion of completeness in traceability changes the way it is perceived, established, maintained, and used. We introduce and discuss the notion of completeness in traceability, which considers *all* *satisfied by* trace links between the requirements and the target artifacts that work to satisfy the requirements, and contrast it with the partial traceability common in practice. The intent of this RE@Next! paper is to highlight the aspect of completeness in traceability and discuss the ramifications of incomplete (or partial) traceability. *Mike: Do we want to go here? I'm leaving it in, but I'm not sure that we need or want this.* While establishing complete traceability may seem extraordinarily difficult to establish in practice [4], our hypothesis is that, given use of formal methods and model based development, it can be achieved in an automated and efficient manner. We have recently investigated *inductive validity cores* [1] as a mechanism to produce sets of support and believe it can be generalized towards all cores.

II. Motivating Example

Mike: Yuck.

Consider a safety device whose requirement is to *raise an alarm if a hazardous condition occurs and is persistent for 5 seconds*. The device is composed of two redundant sensors (for fault tolerance) whose requirement is to *detect the hazardous condition and report if it is persistent for 5 seconds* and a buzzer that *sounds an audible alarm if either one of sensors report the hazard*; *the buzzer shall otherwise remain silent*. Since the sensors are redundant, to establish that the system level requirements are satisfied, one of the sensors and the buzzer is sufficient. Suppose we only capture the trace from the requirement to the requirements of one of the sensors and the buzzer. Now, if there is a change in device requirement to *raise an alarm if the hazardous condition persists for 3 seconds*, the traceability that we have established only helps identify that one sensor's behaviour should be changed. Since the alternate satisfaction is not documented, if the sensor that was modified for the new requirement fails then the requirement is no longer met and may causes safety hazards. Further, an astute reader might have noticed that the buzzer has two requirements of which only the first one contributes to satisfying the system requirement whereas the other does not. If we were to establish all the trace links between the requirements and its realization, one could use it to identify elements of the realization that do not contribute to satisfying any requirement. While we illustrated with a toy example, in practice systems are complex with hundreds of system requirements and numerous components with potentially thousands of component requirements that may induce several ways to satisfy the requirements. Without insight into the all those ways, in our opinion, the results of analysis using such (partial) traceability with be misleading. Unfortunately, neither the completeness of traceability nor its benefits are discussed in the existing literature.

III. Formal Representation of Traceability

In its simplest sense, traceability is capturing the relationship between artifacts. There are three building blocks for traceability - a *source artifact* from which relationship should be established, *the target artifact* to which the source artifact be related and the *trace relationship*, that describes the association between the artifacts. To provide notation, we write $T \vdash e$ to state that a set of target artifacts T is sufficient to establish a traceability relationship $\vdash$ to source element e. For requirements satisfaction traceability, the source artifacts are requirements Δ, the target artifacts Σ are the elements that realize the requirement such as lines of code, design elements, system or world assumptions, or lower-level requirements, and the trace relationship is *satisfies*. Thus, we write $S \models_s r$ for $S \subseteq \Sigma$ and $r \in \Delta$ when set of target elements S satisfies requirement r. We will use S, S' to denote sets of target elements and r, r' to denote requirements. We assume that the satisfaction deduction $\models_s$ is monotonic on the subset relation over Σ, that is, if $S \subset S'$ and $S \models_s r$, then $S' \models_s r$.

Mike: I'm renaming RST_m the set of support SOS; the fewer terms, the better

The monotonicity of the satisfaction relation means that, unless *all* elements of the implementation Σ are required for a proof, there are multiple implementation sets $S \subset S' \subset \ldots \subset \Sigma$ that can satisfy a given requirement r. However, we are primarily interested in *minimal* sets that satisfy r; tracing a requirement to the entire implementation is not particularly enlightening! We call a minimal set of target artifacts the *set of support* for that requirement, and define the *SOS* relation to associate sets of support to requirements.

$$SOS(r, S) \equiv S \models_s r \ \wedge (\forall S' \ . \ S' \subset S \Rightarrow S' \not\models_s r)$$

SOS maps sets of support to a requirement. As mentioned earlier, there could be many sets of support for a requirement. To capture that notion, we define, *all sets of support (ASOS)* for a requirement as an association to all its sets of support.

$$ASOS(r) : \{ \ S \mid (r, S) \in SOS(r, S) \ \}$$

The *ASOS* for a requirement can be envisioned as a matrix in which each row represents a target artifact in Σ, each column represents a set of support for requirement r, and cells contain an 'x' if the target artifact is used in the set of support and is otherwise blank.*Mike: Does this really aid explanation?* The set of *ASOS*-es for all requirements represents the complete traceability of the system. One can envision this as a 3 dimensional traceability matrix, where the third dimension is each requirement.*Mike: again, is this helpful?*

IV. Complete Traceability

Through formal definitions in the previous section we highlight two critical facets – the semantics and completeness – in establishing traceability. In our experience, we found that just being able to trace a requirement to a target artifact that satisfies it, say a line of code, is not useful; a holistic view of how that line of code in conjunction with other related lines of code satisfy the requirement provides meaningful information to perform analysis such as assessing the impact of a change. By defining a trace from requirements to the set

of target artifacts, we advocate that traceability be captured in a way that upholds its semantic rationale. Further, we also found that many analyses that are performed using sets of trace links without a clear picture whether the trace links are adequate for the purpose.*Mike: Which analyses are we talking about here? This seems like a bold, and unsupported claim.* By considering *SOS* trace links and complete traces for each requirement, we can assess analyses related to requirements satisfaction traceability on a proper semantic foundation. In this section, we elaborate on how the semantic foundations in the previous section help us understand, assess, and use traceability precisely.

A. Categorizing the Set of Support

Establishing *ASOS* for a requirement, one gets a clear picture of the all possible ways that requirement is satisfied. This information helps categorize each target artifact into one of the following groups for that requirement. The relationships are illustrated graphically in Figure 1, and explained formally below.

- **MUST** elements - the target artifacts that are present in all the sets of support for a requirement (black triangles in Figure 1).

$$MUST(r) = \bigcap ASOS(r)$$

- **MAY** elements - target artifacts that are used in some, but not all, sets of support (red dots in Figure 1).

$$MAY(r) = (\bigcup ASOS(r)) \setminus MUST(r)$$

- **IRRELEVANT** elements - target artifacts that are not in any of the set of support (green stars in Figure 1).

$$IRR(r) = I \setminus (\bigcup ASOS(r))$$

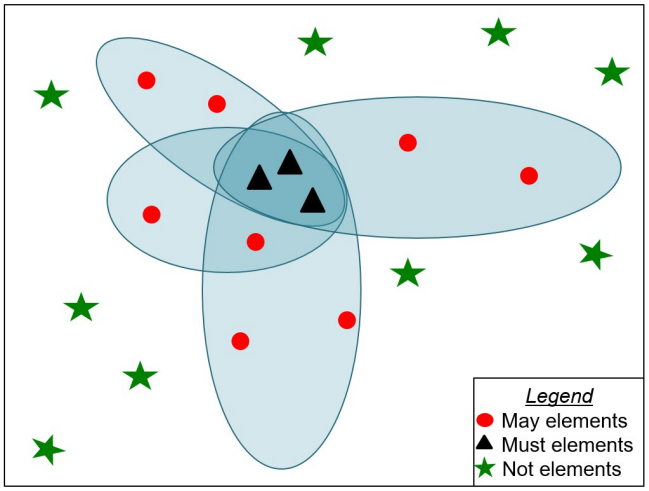

Fig. 1. May - Must - Irrelevant Set of Support

This categorization helps identify the role and relevance of each target artifact in satisfying a requirement. The MUST elements are those target artifacts that are absolutely necessary for the requirement satisfaction. Hence, any change to these elements will most likely impact on each other. On the other hand, MAY elements indicate those target artifacts that satisfy the requirement in one of the possible ways. Any change to just one of these elements will not affect the satisfaction of that requirement. The IRRELEVANT elements are never required to satisfy the requirement, so neither does a change in these artifacts affect the satisfaction of the requirement, nor does a change in the requirement necessitate a change in these elements (at least in terms of satisfaction).

B. Using Traces for Precise Analysis

The categorization of the set of support to be useful in several analyses:

a) Impact Analysis: : The *ASOS* set improves understanding of how a change in the requirement change will affect the target artifacts and vice versa. This categorization precisely helps identify those parts of the system that definitely have to change when there is a change in requirement. MUST elements are those target artifacts that are highly likely to change with any change in the requirement, whereas not all MAY elements may need to be changed. Obviously, one need not worry about the IRR elements. If we examine all the sets of support for all requirements and categorize them, then the elements in the MUST set describe target artifacts that are critical for all requirements *Mike: should we characterize this formally?*.

If we examine changing a target artifact, if it appears in the MUST set for any requirement, then this requirement must be re-verified. However, if it appears in the MAY set for the requirement, then we can instead remove any sets-of-support that contain the element; as long as there still exists at least one set of support for the requirement, no reverification is necessary.

b) Verification and Validation: Complete traceability can assist in tailoring verification and validation in systems. For instance, if several requirements have a certain target artifact in their MUST set, say an particular assumption, it reveals the importance of focusing V&V attention on that artifact. Along the same lines, for a system with a complex architecture (components that each have functionality) such as system of systems, this categorization helps identify components that is critical to satisfy most requirements. This helps plan verification strategies.

Further, the notion of complete traces helps to assess if requirements are satisfied by the system in an unintended manner. It is well known that issues such as vacuity [**?**] can cause requirements to be satisfied in a trivial manner. Even for non-vacuous requirements, it can be the case that requirements can be satisfied using a much smaller portion of the system than intended because they are incorrectly specified. By capturing all the set of support and categorizing them, it is possible to examine whether the MUST set corresponds to expectations on the system, or, if more rigor is required, to examine each set of support individually to see whether it matches expectations.

c) Completeness Checking: By getting all the sets of support for all requirements of the system and categorizing them, it gives a clear picture about the role of target artifacts in satisfying the requirement. By reversing the direction of the complete requirement satisfaction trace, one can find if there are target artifacts that do not trace to any requirement. This can be performed by examining the minimal set of target elements used by *any* SOS for *all* requirements. If so, it is a possible indication of "gold plating" or missing requirements. In other words, it helps assess if there requirements of the system are completely specified with respect to describing all the behaviors of the system. Being able to assess the coverage of requirements over the model is crucial in the safety critical system domain.

d) Comparing Traceability Approaches: There is substantial interest within the Requirements Engineering research community (and our immediate research group) towards automating the construction and maintenance of traceability links. To that end, there are repositories such as *Mike: Name of Jane's repository HERE + citation* containing many example systems, each with a reasonably complete set of requirements and target artifacts and with trace links constructed by groups of experts. It is then possible to benchmark automated and semi-automated traceability approaches against vetted sets of trace links.

We are, in general, very supportive of this approach: benchmarking against a reasonable set of candidate models has been very important in several areas of computer science research, including CPU, GPU, and compiler performance *Mike: citations here*, performance of SAT/SMT solvers and model checkers *Mike: cite SATcomp and SMTcomp*, and many other areas. However, it can also be misleading if the benchmark problems are not carefully selected or if results are incorrectly computed, as described in, e.g., several articles critical of benchmarking for CPU/GPU performance *Mike: citations here*.

For traceability research, the usual standard measures for examining the performance of different approaches is in terms of *precision* and *recall* against the "gold standard" set of traceability links that exist in repositories. Our concern is that, for requirements satisfaction traceability, there are often many such sets of valid links, as we have explored in this paper, so these metrics may be misleading. One can envision situations in which the gold standard pursues one set of support and the automated approach pursues another, leading to low precision and recall scores. A close examination of traceability links and categorizations such as the ones we have explored may be useful to provide more accurate measurements of the quality of automated approaches. If requirements have more than one SOS, the gold standard should probably contain each of these arguments. In addition, the ability to characterize trace links as *MAY* or *MUST* may be a fruitful direction of future research for automated approaches to constructing trace links.

Mike: Stopped here. Not sure we want the discussion below at all. Contrary to the general notion that it is impossible or extraordinarily difficult to identify all trace link in practice [4], some of the initial results from our recent efforts [3] indicate that such complete requirements satisfaction trace can be established, in fact automatically in the realm of formal methods and model based developments. In the next section, we breifely describe our prior work and our plans to extend it to establish complete traceability. While the details the approach and initial results are not in scope of this paper, an interested reader is directed to [3].

Anitha: I am working on the following sections...

V. Discussion

VI. Conclusion

References

[1] E. Ghassabani, A. Gacek, and M. W. Whalen. Efficient Generation of Inductive Validity Cores for Safety Properties. *ArXiv e-prints*, Mar. 2016.

[2] O. C. Z. Gotel and C. W. Finkelstein. An analysis of the requirements traceability problem. In *Requirements Engineering, 1994., Proceedings of the First International Conference on*, pages 94–101, Apr 1994.

[3] E. PAPER. Elaheh paper. In *ELAHEH PAPER*, March 2016.

[4] D. Strašunskas. Traceability in collaborative systems development from lifecycle perspective. In *Proceedings of the 1st International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE02). ACM, New York*, 2002.

[5] P. Zave and M. Jackson. Four dark corners of requirements engineering. *ACM transactions on Software Engineering and Methodology (TOSEM)*, 6(1):1–30, 1997.