Revisiting the Fundamentals of Traceability between Requirements and Their Realizations

Anitha Murugesan, Michael W. Whalen, Elaheh Ghassabani and Mats P.E. 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—what we call *Requirements Satisfaction Traceability*—has been an active research area for some time, none of the literature discusses the fact that there are often multiple ways in which a requirement can be satisfied—there are multiple satisfaction arguments. The distinction between establishing a single satisfaction argument between a requirement and its implementation (tracing one way the requirement is implemented) vs. tracing 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. In this RE@Next! paper, 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.**

I. Introduction

Requirements traceability can be defined as

> *"the ability to describe and follow the life of a requirement, in both 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)."* [9].

This topic has been 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 be 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 called *target artifacts* [8]—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* [20] offer a semantically rich way to establish them. Originally proposed by Zave and Jackson [20], 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 a *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 is the set of 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 a proof. The existing traceability literature does not discuss multiple alternative satisfaction arguments or sets of trace links for one system design.

In our opinion, in the context of satisfaction arguments and satisfies trace links, it is beneficial to distinguish between trace links to *"a"* set of support (containing the clauses needed to make a satisfaction argument) vs. *"the"* sets of support (all clauses needed to make all possible satisfaction arguments).

Establishing trace links to all sets of support, that we call *complete* traceability, provides insight into the elements of the set of support—elements that are necessary for any satisfaction argument, elements that are needed for some satisfaction arguments, and elements that do not contribute to the satisfaction of the requirement of interest. We categorize the elements as *Must*, *May* and *Irrelevant* support elements for each requirement.

In this paper, we examine how this notion of completeness in traceability changes the way traceability is perceived, established, maintained, and used. We introduce and discuss the notion of completeness in traceability, which considers **all** *satisfied by* trace links between a requirement and the target artifacts that work to satisfy the requirement, and contrast this notion 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.

II. Motivating Example

Consider a safety monitoring device that is used to detect and report various hazardous condition in a critical care unit in a hospital. Among the several capabilities of the device, one of its requirement is to *raise an alarm if the room temperature is* *more than* $70°F$ *for more than 5 minutes continuously*. In these systems, establishing traceability between the requirements to parts of the system that satisfy them is crucial to analyze and assess the impact of a change in either the requirements or the system on the other. Say, for this requirement, a trace link is established to a temperature sensor (*measures the room temperature and reports the hazard if it is more than* $70°F$ *for more than 5 minutes continuously*) and, a LED (*produces a blinking red light if the temperature sensor reports the hazard*) – the two components that together satisfy the requirement. This trace helps identify which component gets impacted by a change in the requirement. For instance, if the device is now expected to raise an alarm if the room temperature is more than $80°F$ (instead of $70°F$), using the trace links established one could easily identify that the sensor functionality needs to be changed to meet the new requirement.

However, if there was an alternate way the previous requirement was satisfied, that was not explicitly traced, then inferences from impact analysis is inadequate. If this device had two redundant sensors and different alarming components, say for fault tolerance, then the device might continue to raise an alarm when temperature is more than $70°F$ inspite of the change to one of the sensors to measure $80°F$. Further, if components satisfy more than one requirement, such as the buzzer and LED that can alarm for other types of hazards, being able to trace back to all the requirements those components satisfy, help precisely assess the impact of changes. Hence, without insight into the all the possible trace links between requirements and its realization, in our opinion, the results of such analysis using traceability could be potentially inadequate and misleading. While we explained the problem of incompleteness in traceability 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. 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 $S \vdash_s r$ 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 $\vdash_s$ is monotonic on the subset relation over Σ, that is, if $S \subset S'$ and $S \vdash_s r$, then $S' \vdash_s r$.

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.

$$SOS(r,S) \equiv S \vdash_s r \;\wedge (\forall S' \,.\, S' \subset S \Rightarrow S' \nvdash_s r)$$

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 set of *ASOS*-es for all requirements represents the complete traceability of the system.

IV. Complete Traceability

Through formal definitions in the previous section we highlight two critical facets – the semantics and completeness – in establishing traceability. In practice, just being able to trace a requirement to a target artifact that satisfies it, say a line of code, is not useful [10]; 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 [13]. 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 performing analysis, such as impact analysis, using a partial set of trace links may result in imprecise results and misplaced confidence about the system. 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).

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

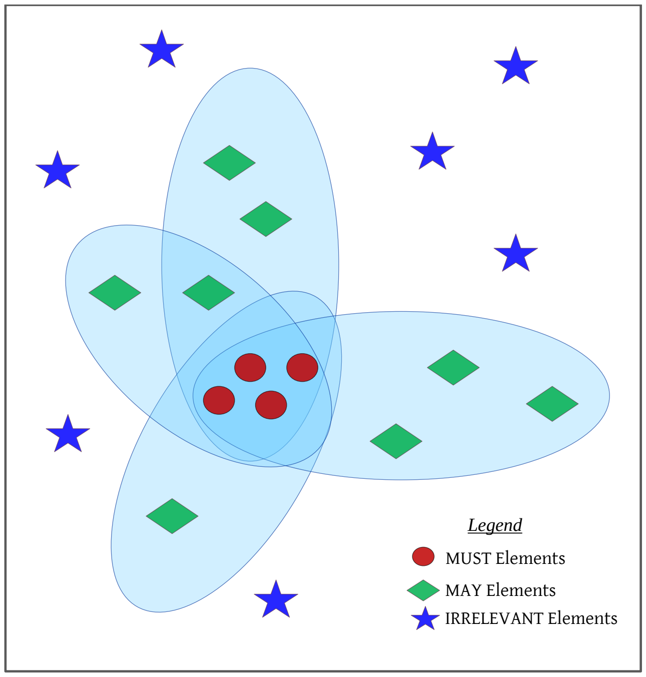

Fig. 1. May - Must - Irrelevant Set of Support

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

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

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 will affect the target artifacts and vice versa. While the *ASOS* of a requirement gives a clear picture of various ways a requirement is satisfied by the system, the categorization of target artifacts helps precisely assess and plan when and where the changes have to be implemented. The *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.

If a requirement has elements only in its *MAY* set, that is if *MUST* set is empty ($MUST(r) = \varnothing$), it indicates that the requirement has been (intentionally or unintentionally) implemented in independent ways, such as fault tolerant systems. For such requirements, one has to carefully analyse and decide if the target artifacts in all or one disjoint set needs to be changed. These analysis could be performed either from the perspective of one or all requirements of the system.

From the target artifact side, this categorization helps analyse their *Anitha: degree of freedom?* to change. Say, we decide to change a target artifact in the *MAY* set for a requirement. While one might think that it is safe to change this artifact since it does not affect that requirement's satisfaction, an examination of the *ASOS* sets of other requirements helps identify if it is indeed safe to change that artifact. If it is present in the *MUST* set for another requirement, then a change to this artifact will definitely impact the other requirement. However, if it is in the *MAY* sets for all the requirements, then it is clearly safe to change. Hence, this categorization helps us to precisely assess critical dependencies between the target artifacts and the satisfaction of requirements and thus enables a precise bi-directional impact analysis of a change.

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.

If we examine changing a target artifact that appears *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 for that requirement.

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 [14] 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, one can get a clear picture about the role of target artifacts in satisfying the requirement(s). 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, in other words the $\{\forall r \in R \bigcap IRR(r)\}$. 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) Benchmarking in Traceability: There is substantial interest within the Requirements Engineering research community (and our immediate research group) towards automating the construction and maintenance of traceability links [11], [5], [4]. To that end, there are repositories such as the Data sets published at Center of Excellence for Software Traceability [1] 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 [12], GPU [19], and compiler performance [3], performance of SAT/SMT solvers and model checkers [18], 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 [6], [2], [16].

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.

V. Discussion

While the notion of correctness and completeness in requirements traceability is not a new topic of discussion, our argument is that it is not been rigorously defined and assessed. The intent of this RE@Next! paper is to clarify the foundations of tracing requirements to its realization - in particular the requirements satisfaction traceability – and explaining its the benefits and ramifications.

While there are enormous amount work in the area of requirements traceability, there are no, to the best of our knowledge, rigours yardsticks to quantitatively and qualitatively access the trace links. This raises a qualm about the way traceability is assessed. Hence, we believe that there is a strong need to identify metrics, define approaches and develop tools that will rigorously assess trace links.

In this paper we define a strong theoretical framework to understand and assess requirements satisfaction traceability, that we believe is an initial attempt towards addressing the above need. By defining a requirements satisfaction trace from a requirement to a set of support, rather than a target artifact, we intertwine the rationale to verify/validate its correctness automatically. Such traces are self-sufficient to be verified for correctness, without the need for another source of truth or spending enormous amount of time to validate them. By examining these traces one could verify and validate if the system indeed satisfies the requirements in s intended ways. Further, this helps redefine the notion of completeness in traceability from capturing trace links to all requirements to capturing *all* trace links to all requirements. It is worth reiterating that our goal is not yet another formalization of traceability but to provide the foundation for any future research on satisfaction trace link generation and assessment.

Contrary to the general notion that it is impossible or extraordinarily difficult to identify all trace link in practice [17], some of the initial results from our recent efforts [7] indicate that such complete requirements satisfaction trace can be established, in fact automatically in the realm of formal methods and model based developments. Previously [15], we demonstrated a model based approach to system construction in which compositional proofs are used to automatically establish satisfaction arguments. The approach was based on mathematically proving the requirements with respect to a model of the system. As an extension of that, we recently developed a technique [7] to extract the elements of the proof, that we call *inductive validity cores (IVC)*, to present an explanation to how the requirements were verified in the model. These IVCs are nothing but the set of support for a requirement. Our next step is to use this approach iteratively to extract all the sets of support for all the requirements and establish a complete traceability for the system. While the details the approach and initial results are not in scope of this paper, an interested reader is directed to [7].

References

[1] Center of Excellence for Software Traceability. http://www.coest.org, 2016.

[2] AnandTech. They’re (almost) all dirty: The state of cheating in android benchmarks. http://www.anandtech.com/show/7384/state-of-cheating-in-android-benchmarks, 2013.

[3] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, et al. The nas parallel benchmarkssummary and preliminary results. In *Proceedings of the 1991 ACM/IEEE conference on Supercomputing*, pages 158–165. ACM, 1991.

[4] J. Cleland-Huang, B. Berenbach, S. Clark, R. Settimi, and E. Romanova. Best practices for automated traceability. *Computer*, (6):27–35, 2007.

[5] A. Egyed and P. Grunbacher. Automating requirements traceability: Beyond the record & replay paradigm. In *Automated Software Engineering, 2002. Proceedings. ASE 2002. 17th IEEE International Conference on*, pages 163–171. IEEE, 2002.

[6] ExtremeTech. Driver irregularities may inflate nvidia benchmarks. http://www.extremetech.com/computing/54154-driver-irregularities-may-inflate-nvidia-benchmarks, 2003.

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

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

[9] 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.

[10] J. Guo, N. Monaikul, and J. Cleland-Huang. Trace links explained: An automated approach for generating rationales. In *Requirements Engineering Conference (RE), 2015 IEEE 23rd International*, pages 202–207. IEEE, 2015.

[11] J. H. Hayes, A. Dekhtyar, and J. Osborne. Improving requirements tracing via information retrieval. In *Requirements Engineering Conference, 2003. Proceedings. 11th IEEE International*, pages 138–147. IEEE, 2003.

[12] J. L. Henning. Spec cpu2006 benchmark descriptions. *ACM SIGARCH Computer Architecture News*, 34(4):1–17, 2006.

[13] E. Hull, K. Jackson, and J. Dick. *Requirements engineering*. Springer Science & Business Media, 2010.

[14] O. Kupferman and M. Y. Vardi. Vacuity detection in temporal model checking. *International Journal on Software Tools for Technology Transfer*, 4(2):224–233, 2003.

[15] A. Murugesan, M. W. Whalen, S. Rayadurgam, and M. P. Heimdahl. Compositional verification of a medical device system. In *ACM Int'l Conf. on High Integrity Language Technology (HILT) 2013*. ACM, November 2013.

[16] M. Sayeed, H. Bae, Y. Zheng, B. Armstrong, R. Eigenmann, and F. Saied. Measuring high-performance computing with real applications. *Computing in Science and Engineering*, 10(4):60–70, 2008.

[17] 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.

[18] M. N. Velev. Superscalar suite 1.0. http://www.ece.cmu.edu/ mvelev, 1999.

[19] V. Volkov and J. W. Demmel. Benchmarking gpus to tune dense linear algebra. In *High Performance Computing, Networking, Storage and Analysis, 2008. SC 2008. International Conference for*, pages 1–11. IEEE, 2008.

[20] P. Zave and M. Jackson. Four dark corners of requirements engineering. *ACM transactions on Software Engineering and Methodology*, 1997.