Achieving Lightweight Trustworthy Traceability

Jane Cleland-Huang and Mona Rahimi  
Systems and Requirements Engineering Center  
School of Computing, DePaul University  
Chicago, IL USA  
jhuang@cs.depaul.edu,  
m.rahimi@acm.org

Patrick Mäder  
Technische Universität Ilmenau  
Software Systems Group  
Ilmenau, Germany  
patrick.maeder@tu-ilmenau.de

ABSTRACT

Despite the fact that traceability is a required element of almost all safety-critical software development processes, the trace data is often incomplete, inaccurate, redundant, conflicting, and outdated. As a result, it is neither trusted nor trustworthy. In this vision paper we propose a philosophical change in the traceability landscape which transforms traceability from a heavy-weight process producing untrusted trace links, to a light-weight results-oriented trustworthy solution. Current traceability practices which retard agility are cast away and replaced with a disciplined, just-in-time approach. The novelty of our solution lies in a clear separation of trusted trace links from untrusted ones, the change in perspective from ‘living-with’ inaccurate traces toward rigorous and ongoing debridement of stale links from the trusted pool, and the notion of synthesizing available ‘project exhaust’ as evidence to systematically construct or reconstruct purpose, highly-focused trace links.

Categories and Subject Descriptors
D.2.1 [Software Engineering]: Requirements/Specifications

General Terms
Human Factors, Documentation

Keywords
Traceability, Safety Critical

1. WHY TRACEABILITY?

In his rather infamous argument against the US Reagan-era missile defense program, David Parnas contended that the planned star-wars program was inherently flawed because it would never actually work [2]. His argument had two major prongs. First, the complexity of the system (at that time) would make it difficult, if not impossible, to build correctly; and secondly, that as nobody would actually trust it, it would not lead to any reduction in the overall weapons buildup. These two issues are strikingly similar to the challenges found in the current traceability landscape.

Defined by the Center of Excellence for Software Traceability (CoEST) as “the ability to interrelate any uniquely identifiable software engineering artifact to any other, maintain required links over time, and use the resulting network to answer questions of both the software product and its development process” [7], traceability has proven to be quite difficult to attain and maintain in practice [10]. The complexity is not so much technical as process-related; because creating and maintaining trace links accurately and consistently requires complex, tool-supported project-wide choreography, which typically fails [14, 12].

On the other hand, the importance of traceability, especially in safety-critical systems, is well understood. Numerous certifying bodies prescribe a set of artifacts to be created and trace links to be established. For example, the USA Federal Aviation Administration (FAA) expects manufacturers of airborne systems to establish “traceability between source code and low-level requirements” in order to “enable verification of the absence of undocumented source code and verification of the complete implementation of the low-level requirements.” Similarly, the USA Food and Drug Administration (FDA) states that traceability analysis must be used to verify that the software design implements the specified software requirements, that all aspects of the design are traceable to software requirements, and that all code is linked to established specifications and test procedures [9].

Furthermore, the emerging trend towards the adoption of agile methods in regulated domains introduces the need for more continual evaluation. In fact, the European Open-DO initiative [13], actively seeks to address the Big Freeze problem in which the significant cost and effort of the certification process make it difficult for change to be introduced once the product is certified. The Open-DO initiative has a specific goal to develop techniques and tools that will allow all software to be constantly maintained in a ‘ready-to-certify’ state.

2. TRACEABILITY FAILURE

Unfortunately, in practice, trace links are typically incomplete and inaccurate. Our prior study of several different safety critical projects, showed that traceability data exhibit numerous flaws including missing artifact types, missing trace paths, redundant and/or ambiguous trace paths, and numerous missing trace links [14, 12]. As a result, traceability data is undervalued, untrusted, and underutilized. We therefore question why something that is so valued by
safety-critical regulatory and certification agencies is so dys-
function in practice, and furthermore, what can be done
about it?

One of the underlying root causes is the fact that while
traceability is designed to support change impact analysis,
in practice it can retard progress by adding additional over-
heads to the maintenance process. Any modifications made
to core artifacts, such as requirements, design, or code, intro-
duce the overhead of requiring corresponding updates to the
traceability data. Missing this second step leads to degrada-
tion of the trace links. It is therefore no wonder that agile
developers denigrate “traceability” to the vilified heavy-weight
category.

This is far from a new problem. As far back as 1995, Gotel
et al. identified several different traceability problems and
attributed them to coordination problems, lack of perceived
benefits, time to market pressures, and lack of ownership
[10]. These problems observed almost 20 years ago, continue
to plague the traceability landscape today, meaning that the
traceability gap between what is prescribed and what is prac-
ticed is still very real [14, 12].

3. CAN WE FIX IT?

It is our position that the underlying philosophy and prac-
tices of traceability as prescribed and practiced today are
fundamentally flawed and inherently unfixable. In this vi-
sion paper we therefore propose a philosophical change
in the traceability landscape which transforms traceabil-
ity from a heavy-weight process to a light-weight results-
oriented solution. Current traceability practices which weigh
down development are cast aside and replaced with a disci-
plined just-in-time approach. The crux of our solution
lies in a clear separation of trusted trace links from
untrusted ones, the constant debridement of stale links from
the pool of trusted links, and a collection of untrusted
trace link evidence which is used to systematically con-
struct or reconstruct purposed trace links in support of
highly focused tasks such as that of building a safety or as-
surance case. Trace link evidence is derived from numerous
sources including, but not limited to, debrided trace links,
trace retrieval results [8], version control logs [3], and poten-
tially monitored development environments [1]. The idea
is that absolutely nothing produced as a byproduct of the
development process is wasted; instead every single piece of
project exhaust is saved and then utilized in a remediation
process in order to promote untrusted links back into the
trusted domain.

The primary benefits of this approach are (1) that users
will always know which trace links have been vetted by hu-
mans and can be trusted, (2) untrusted, yet useful, trace
information will be made available to users for trace con-
struction and/or reconstruction purposes, (3) human ana-
lysts will no longer be obligated to maintain links during the
engineering process – although any links that are created and
vetted will be placed into the trusted set and monitored for
change impact, and finally (4) only trace links that are actu-
ally needed will be created and/or maintained – thereby the
cost of traceability will always be clearly offset by its ben-
efits. While the philosophy of just-in-time development is
certainly not new, it has not yet effectively been integrated
into the traceability process. We have however seen similar
trends pervade other areas of Software Engineering such as
requirements engineering and software architecture. In these
areas, commonly accepted practices that previously relied on
upfront elicitation, specification, and design activities have
been progressively replaced with more agile just-in-time ap-
proaches.

4. TRACE LINK DEBRIDEMENT

Our proposed separation of links into trusted and un-
trusted bases is dependent upon the ability to successfully
recognize when links should be demoted to the untrusted
state. In the medical field, the term debridement refers to
the act of removing dead flesh caused by a complex wound.
While it can be performed surgically, there has been a re-
cent upsurge in the use of maggots – which apparently have
an inclination to eat dead, rather than live, tissue. In the
traceability domain, stale trace links are akin to dead flesh,
and algorithms capable of recognizing and removing stale
links are akin to the maggots.

The debridement practice is quite simple. If a link be-
comes suspect, it is immediately removed from the trusted

Figure 1: Trusted Link Model. Any link which becomes suspect is moved to the untrusted base. Untrusted
links are then used in conjunction with link evolution patterns, trace retrieval results, and link evidence, to
guide a user through a trace remediation process in order to construct just-in-time trace links.
set and placed, with rationales and supporting data, into the untrusted set. This data, which is preferably annotated with probabilistic information derived from historical analysis of modifications and link likelihoods, will later be used for link reconstruction purposes.

Unfortunately, we currently do not have effective trace debride ment algorithms available. The current state of practice for recognizing a suspect link, is rather coarse-grained, as a link is marked as suspect if either the source or target artifacts of the link are modified in any way. In practice, this produces trace matrices in which a high percentage of links are marked as suspect, and would in our approach ultimately produce a rather small set of trusted links. Nevertheless, separating trusted and untrusted links into separate pools is a significant improvement on the current state of affairs in which there are hundreds, or even thousands, of trace links – all of which are untrusted.

Moving forward, we need to develop effective debridement algorithms which understand the impact of change upon trace links in order to recognize and debride untrusted links when modifications occur in order to constantly maintain the trusted base in an accurate, incomplete state. Furthermore, when debridement occurs – we need to understand exactly what supporting information should be retained to facilitate later link reconstruction. Therefore, future research directions should explore which change scenarios lead to trace decay, and under which conditions changes do not trigger decay and can be safely ignored.

In Figure 2 we provide two simple examples of change scenario patterns but make no attempt to present a complete set in this vision paper. In the baseline case a link exists from class C to requirement R. Furthermore, R is marked by the user as satisfied – meaning that not only is the link trusted, but also that the single class C is sufficient to satisfy R.

In the first scenario, a method m in class C is promoted to a new class Cm. As a result, the original link becomes suspect as it is unknown whether the link should be transferred to Cm or retained by C’ (the modified version of C). In traditional systems the only information retained would be that the single trace link from C to R was suspect. In our new proposed approach, we automatically add the new link from Cm to R, mark both links as suspect, demote them to the untrusted set, and document a set of remedial actions and an associated rationale. In this case remediation involves determining whether to retain only the original link, duplicate the link, or replace the original link with a link from Cm to R. Furthermore, in addition to saving the (now) untrusted trace link data, we also save additional information which can later be used to reconstruct trusted trace information.

In the second scenario, a method in class C is deleted, producing a modified class C’. The trace link from C to R is no longer trusted, and it is unknown whether R remains satisfied. Remediation efforts include evaluating the link from C’ to R to determine whether it is still needed, and evaluating the satisfaction of R. If unsatisfied, and if R is found to be relevant, then a form of technical debt is incurred.

To this end, it is clear that we need to recognize patterns of change and to create a trace link change calculus. While very limited prior work exists in this area, it has focused on highly specific contexts such as evolution of requirements [4, 6] and of UML class diagrams [11]. Such approaches need to be generalized so that they can be applied across the entire life-cycle, and used to (1) recognize the need for debridement, (2) build rigorous and systematic trace link remediation processes, and (3) enhance the remediation process with statistical models that guide the ordering of information according to likelihoods and then prune alternative options based on early user decisions. For example, if a given change means that either link L1 is correct, or link L2, then we present the most likely link first, and if approved by the user, remove the other link from consideration. Constructing such trace link models will require significant research effort, but will ultimately support the kind of light-weight traceability we envision for the future.

4.1 Trace Reconstruction Decisions

In our model the developer and/or analyst can choose whether to resolve a suspect link at the time of change, or whether to defer this decision for later. In other words, our model makes it acceptable to defer the remediation process, based on the notion that it is often more cost effective to store the information needed to remediate a link only if the link is actually needed to perform a software engineering task. There are clear trade-offs. Remediating a link immediately following the initial change means that the developer has the necessary knowledge for the remediation process fresh in his or her mind; however, on the other hand, practice has repeatedly shown that most developers fail to
perform this task. Our approach acknowledges this fact, saves pertinent information, thereby supporting deferral of the trace link remediation process. A side-effect of our philosophy is that we no longer think in terms of trace matrices such as the one depicted in Figure 3, which are constructed systematically as the life-cycle proceeds. Trace matrices are less meaningful in the world of just-in-time traceability, as they are inevitably incomplete much of the time. Instead, we think in terms of task-specific trace slices or trace paths, such as the one depicted in Figure 4, which depicts the trace slice generated to demonstrate that a specific hazard $H$ has been fully mitigated in the delivered system. Certainly matrices can be transformed into slices, and vice versa, but in our approach, it is the trace slice rather than the matrix which is seen as the fundamental building block.

5. CONCLUSION

In this vision paper, we have drawn attention to the current state of the practice in which traceability guidelines prescribed by various regulatory bodies are simply not delivered consistently by manufacturers. We have proposed a solution which recognizes the human component of the traceability process, clearly separates trusted from untrusted links, and systematically collates trace link evidence for use in the remediation process. This process for collecting and collating evidence for a link can also be applied in a cold-start scenario in which trace links are being created for the first time. The greatest advantage of our approach is the fact that the costs of traceability are incurred only when the benefits can be immediately realized. As a vision paper, we have not yet constructed the trace link calculus or developed heuristics for demoting trusted links to untrusted ones, or for recreating trusted links from an untrusted pool of evidence. These problems belong to a class of trace maintenance challenges which has traditionally received far less attention than trace creation problems [5]. However, these challenges represent pressing issues that must be addressed in order to close the traceability gap between what is prescribed by certifying bodies and what is actually delivered.

6. ACKNOWLEDGMENTS

The work in this paper was partially funded by the US National Science Foundation Grant CCF:1319680 and by the German Ministry of Education and Research (BMBF) grants: 16V0116, 01IS14026A.

7. REFERENCES