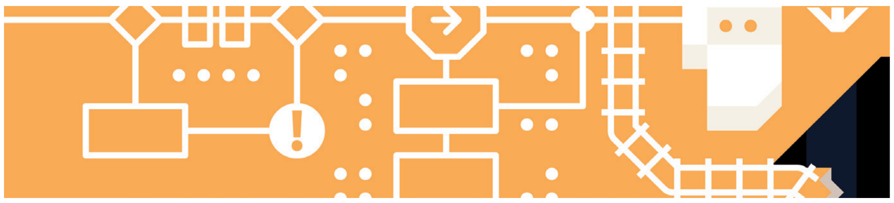
Strategic Traceability for Safety-Critical Projects

Patrick Mäder, Ilmenau Technical University

Paul L. Jones and Yi Zhang, US Food and Drug Administration

Jane Cleland-Huang, DePaul University

// *An evaluation of traceability information for 10 submissions prepared by manufacturers for review at the US Food and Drug Administration identifies nine widespread traceability problems that affected regulators' ability to evaluate products safety in a timely manner.* //



FAILURE OF SAFETY-CRITICAL software systems to operate correctly can cause serious harm to the public—consider devices such as pacemakers, nuclear power systems, and train signals, all of which run on safety-critical software. Therefore, teams building safety-critical software products must perform rigorous risk analyses to identify potentially unsafe conditions and their contributing factors. Many projects conduct this process using techniques such as failure modes and effects analysis, fault tree analysis, and hazard and operability studies. The risk analysis produces a set of system-level requirements specifically designed to mitigate or eliminate faults and reduce the likelihood of accidents.[1] These requirements relate to a broad range of factors including training, testing, process improvements, hardware, human factors, and software design constraints.

In this article, we focus on traceability's role in establishing evidence that device specifications and implementations address identified hazards and their risk control measures (see the "Traceability Standards in Safety-Critical Projects" sidebar).[2] Creating and maintaining trace links can be an arduous, error-prone, and costly process that can have a significant effect on the overall costs and time-to-market for a product.[3–5] Traceability practices, therefore, need to be strategically planned and carefully implemented to provide cost-effective support for evaluating and demonstrating a specific system's safety and security.[6] When traceability isn't implemented strategically, individual stakeholders might create traces that they personally consider to be important or attempt to provide complete trace coverage without considering how the resulting trace links will be used. A brute-force approach to traceability has been shown in practice to be difficult to implement, almost impossible to maintain, and not particularly helpful for providing evidence that a system or device is safe for its intended use.

We present six practices for strategic traceability, derived from our own observations of effective traceability in industrial projects and supported by current literature.[3–6] We also identify nine recurring problems, each of which reduces the effectiveness of traceability verification efforts and increases the difficulty experienced by regulators in evaluating product safety. All the observations in this article are based on actual observations, but the illustrative examples are either fictitious or built on obfuscated data.

Effective Practices for Tracing in Safety-Critical Projects

Although all the cases reported in this article are safety-critical in nature, many of the problems that we discuss are also applicable to software and

0740-7459/13/$31.00 © 2013 IEEE

TRACEABILITY STANDARDS IN SAFETY-CRITICAL PROJECTS

Traceability is an established tenet in the software engineering community and is essential for assuring that software is safe for use. Many regulatory agencies of various industry sectors have recognized its importance and have subsequently incorporated it into various standards and guidelines. For example, the Federal Aviation Administration DO-178C standard specifies that at each stage of development, "software developers must be able to demonstrate traceability of designs against requirements."[1] The automotive safety standard ISO 26262:2011 dedicates an entire section to requirements management and states, for example, stating that "safety requirements shall be traceable … to: each source of a safety requirement at the upper hierarchical level, each derived safety requirement at a lower hierarchical level, or to its realization in the design, and the specification of verification."[2]

The ANSI/AAMI/IEC 62304:2006 standard addresses the medical device software development life-cycle processes, requiring "traceability between system requirements, software requirements, software system test, and risk control measures implemented in the software" and that "the manufacturer shall verify and document that the software requirements are traceable to the system requirements or other source."[3] Similarly, the US Food and Drug Administration (FDA) states that traceability analysis must be used to verify that the software design of a medical device 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.[4] Fergal Mc Caffery and his colleagues provide a comprehensive discussion of traceability requirements for medical device software development.[5]

References

1. *DO-178C/ED-12C, Software Considerations in Airborne Systems and Equipment Certification*, RTCA, 2011.
2. *ISO DIS 26262:2011, Road Vehicles—Functional Safety, International Organization for Standardization*, 2011.
3. *ANSI/AAMI/IEC 62304:2006, Medical Device Software—Software Life Cycle Processes*, Assoc. Advancement Medical Instrumentation, 2006.
4. *Guidance for the Content of Premarket Submissions for Software Contained in Medical Devices*, US Food and Drug Administration, 2005.
5. F. Mc Caffery et al., "Medical Device Software Traceability," *Software and Systems Traceability*, J. Cleland-Huang, O. Gotel, and Andrea Zisman, eds., Springer, 2011, pp. 321–339.

systems traceability in general product development efforts. The following practices can be used to establish traceability that's cost-effective and that provides effective support for constructing a safety-critical system and assessing its safety. We present the practices in the order in which we might expect them to be adopted. In some cases, higher-level practices are dependent on lower-level ones.

Practice 1: Plan Your Traceability

Project managers should strategically plan traceability in a project's early phases and document it using a traceability information model (TIM).[7] A TIM models the traceable artifact types (requirements, design, code, and so on) and their permitted trace links as a Unified Modeling Language (UML) class diagram. Figure 1 depicts a TIM for a safety-critical project. Artifact

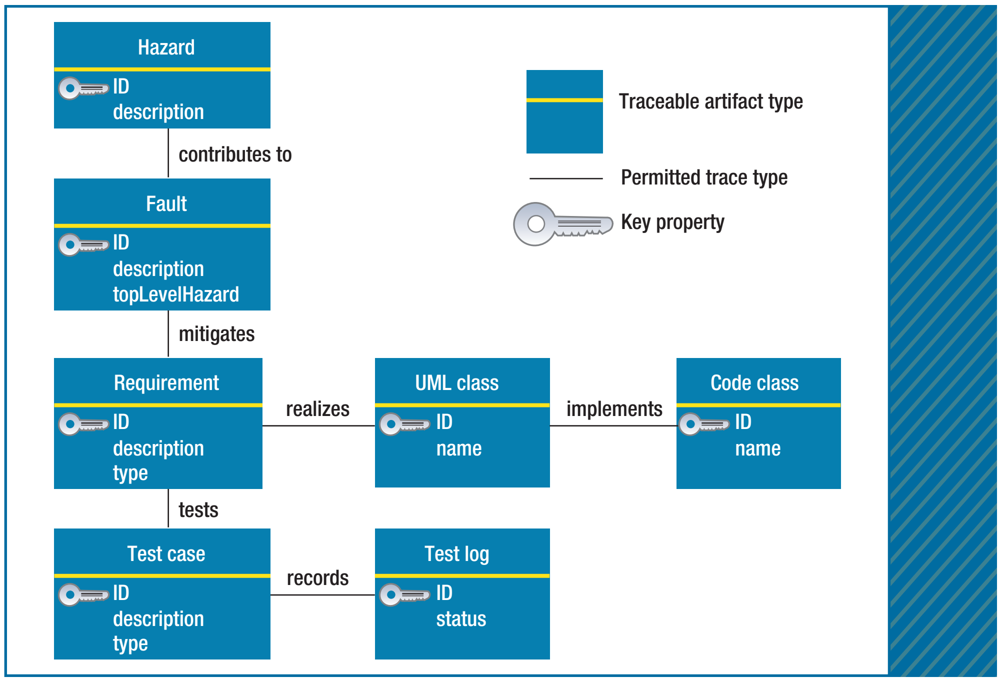


FIGURE 1. A typical traceability information model (TIM) for a safety-critical system. The TIM depicts the planned trace paths among development life-cycle artifacts such as hazards, requirements, and design.

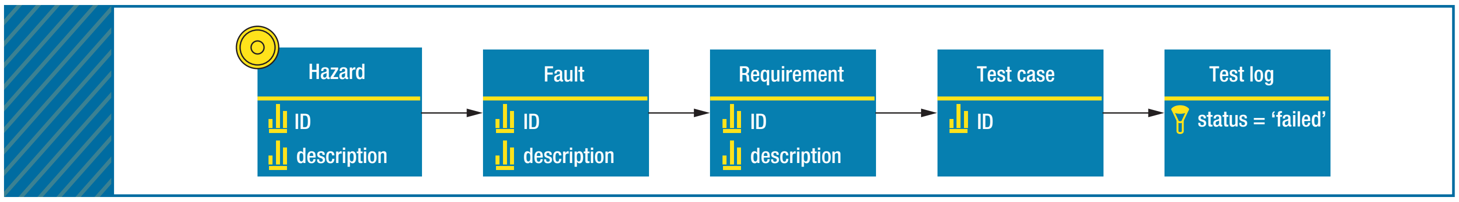


FIGURE 2. A Visual Trace Modeling Language (VTML) query modeled over the TIM in Figure 1. The query retrieves, for a given hazard, the ID and description of related faults and requirements that have assigned failed test cases.

types include hazards, faults, requirements, UML classes, code classes, test cases, and a test log. Trace links are only permitted along the specified paths. For example, requirements can be traced to faults and UML classes to requirements. Furthermore, each traceable artifact type is characterized by one or more properties, such as ID, description, or type, used to generate trace queries and that might also be included in trace query results.

Practice 2: Offer Traceability Tool Support

Creating, maintaining, and using trace links can be time-consuming and arduous. Tracing should therefore be supported using any tool, such as Rational DOORS or Rational RequisitePro, that provides features for establishing, maintaining, and navigating trace links and has the ability to display trace information in formats such as matrices or trace slices. The project environment can also be instrumented to include semiautomated approaches that use information retrieval methods to dynamically generate candidate trace links[5,6] or to infer relationships by analyzing change management systems' commit logs.

Practice 3: Create Traces Incrementally

In practice, the task of creating, evaluating, and approving traceability links is frequently deferred until very late in the project, at which point it's often conducted by people other than the original developers, testers, and requirements engineers. Consequently, trace links are often incomplete and inaccurate[3,6] and aren't available throughout the project to support development. Instrumenting the environment with tracing tools empowers knowledgeable project stakeholders to create trace links incrementally within the context of their daily work. This reduces the likelihood that trace links will be created solely for approval purposes and allows project stakeholders to benefit from traceability knowledge throughout the project.

Practice 4: Model Traceability Queries

Traceability queries cover basic lifecycle activities such as finding all requirements associated with currently failed test cases or listing all mitigating requirements associated with a given hazard. Trace queries can be defined in several ways, for example, by using the Visual Trace Modeling Language (VTML), which represents queries as a set of filters applied to the TIM.[7] These filters eliminate unwanted artifacts and define data to be returned by the trace query. Figure 2 shows a VTML query.

Practice 5: Visualize Trace Slices

In safety-critical systems, trace links established among hazards, faults, mitigating requirements, design, implementations, and test cases are of particular importance.[8] Therefore, instead of presenting traceability material in the form of trace matrices, generate trace slice visualizations in which the hazard is the root node and all direct and indirectly traced artifacts that contribute to mitigating the hazard are shown as a tree. Figure 3 illustrates this with a trace slice for one specific hazard. These slices support safety-related tasks such as helping a regulator to understand how a specific hazard has been addressed in the final system.

Practice 6: Evaluate Traces Continually

One challenge of implementing a traceability process is that the people performing the tracing tasks often don't directly realize tracing benefits. Furthermore, the current status of the traceability effort is often not visible to individual stakeholders or the project manager. A dashboard that displays the tracing progress for a project can be effective for tracking and managing the project's tracing goals and also for motivating team members to create appropriate trace links. The dashboard can display useful information such as burn down charts showing the percentage of hazards that don't have mitigating requirements, or the percentage of mitigating requirements without passed test cases. This information is generated via trace queries. Personalized views can be created for individual project members.

Observed Traceability Problems

Unfortunately, current traceability practices often fall far short of accepted principles in software engineering for developing safety-critical systems. Our two US Food and Drug Administration (FDA) research team members systematically evaluated the traceability documentation presented in 10 submissions

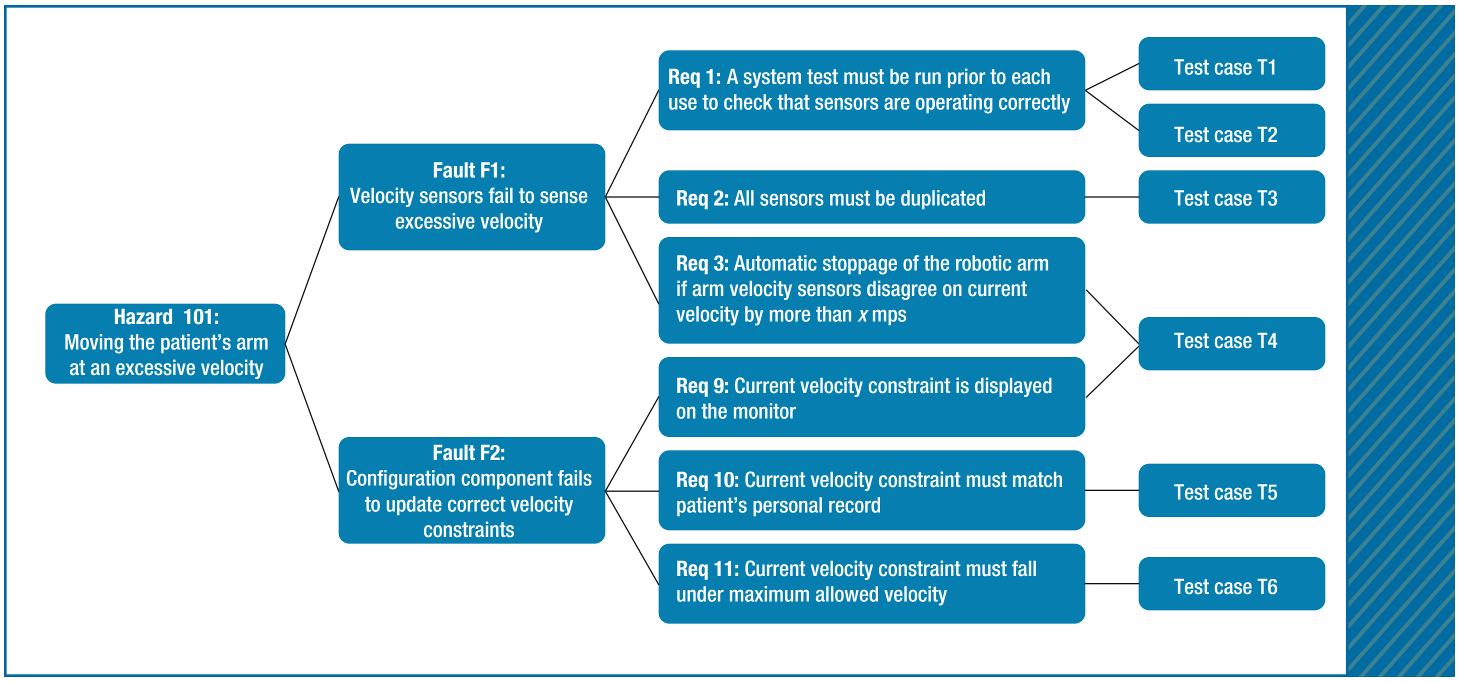

Hazard 101: Moving the patient's arm at an excessive velocity
Fault F1: Velocity sensors fail to sense excessive velocity
Fault F2: Configuration component fails to update correct velocity constraints
Req 1: A system test must be run prior to each use to check that sensors are operating correctly
Req 2: All sensors must be duplicated
Req 3: Automatic stoppage of the robotic arm if arm velocity sensors disagree on current velocity by more than x mps
Req 9: Current velocity constraint is displayed on the monitor
Req 10: Current velocity constraint must match patient's personal record
Req 11: Current velocity constraint must fall under maximum allowed velocity
Test case T1
Test case T2
Test case T3
Test case T4
Test case T5
Test case T6

FIGURE 3. A simplistic trace slice showing test cases, requirements, and faults associated with a hazard. The information can be retrieved via the VTML query from Figure 2. (Example taken from a therapeutic robotic arm case study.[9])

for FDA medical device approval. Their analysis of submissions revealed several issues from which we identified nine distinct problems, specified in terms of definition, trace instance, and presentation problems.

Definition Problems

We identified three types of problems in the area of defining trace strategies. Definition problems appeared to have far-reaching impacts on the tracing process and resulted in ad hoc approaches to traceability and uneven, inconsistent coverage among design artifacts.

Problem 1: Failure to explicitly model a TIM. Without an explicit and documented trace strategy, developers often expend valuable effort in the wrong places while important traces required for demonstrating or arguing product safety are missing.

The lack of traceability planning introduces numerous issues, such as the problem depicted in Figure 4 in which traces are established directly from design requirements to hazards without any intermediate artifacts. This results in high fan-in—for example, in one case we found 15 requirements mitigating a single hazard—and makes it difficult to understand why a particular requirement is linked to a hazard. A better solution would be to decompose the hazards into contributing factors derived from the initial risk analysis, then to create trace links from risk control measures to contributing factors and contributing factors to hazards.

Remedy 1.1. Create a TIM early in the project and assign responsibility to the project manager to ensure that it's followed consistently throughout the development process.

Remedy 1.2. Use the TIM to specify how links will be created. To reduce effort, create manual links only for critical requirements, and address other traceability needs through automated techniques,[10] which use information retrieval methods to generate traces on a just-in-time basis.

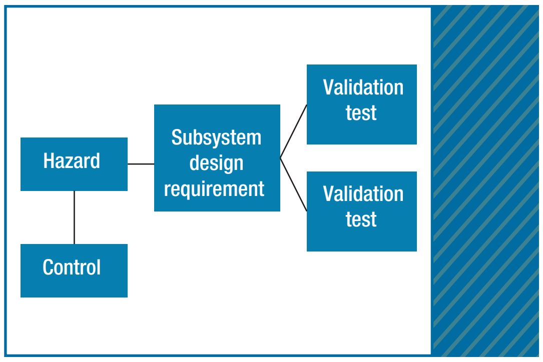


FIGURE 4. A TIM that shows a trace path directly from requirements to hazards, missing the important intermediate step of tracing through contributing faults. In this example, hazards were traced directly up to 15 subsystem design requirements in the worst case.

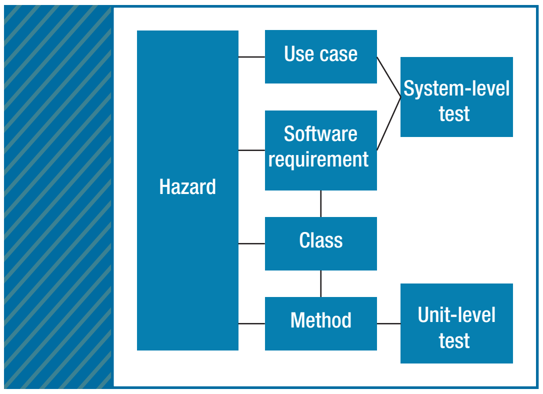


FIGURE 5. A TIM that shows redundant trace paths between sets of artifacts. This TIM suffers from multiple granularity problems.

Problem 2: Trace granularity not clearly defined. Ill-defined trace granularity leads to unacceptably high, unacceptably low, or mismatched links, making it difficult to determine whether hazards and faults are fully addressed. Our study identified three different kinds of granularity issues:

- *Trace links that are too coarse-grained*. The links don't reference artifacts of interest, but instead reference higher-level artifacts—for example, links established between large abstract sections of the requirements specification and test cases. Traces are too coarse-grained when they don't link the test case to the specific requirement being addressed.
- *Trace links that are too fine-grained*. The links reference low-level parts of an artifact, distracting the stakeholder from the fact that the relation to the higher-level, more coarse-grained artifact is important—for example, we observed multiple individual trace links between a requirement and every single step of a test procedure, introducing unnecessary effort to create and maintain so many links. In this case, the entire set of individual links should be replaced by a single link to the test case.
- *Inconsistent granularity*. A set of trace links are inconsistent in the level of artifacts being traced—for example, we observed a single trace matrix, which includes whole sections of the software requirements specification, traced to a single test case, and then, in the same matrix, trace links from individual requirements to multiple test suites. The mismatch of granularity in the trace matrix makes any degree of automated reasoning concerning test coverage very difficult and also complicates the human reviewer's task. (Defining inconsistent granularity at the link level contributes to the inconsistent link problem described in Problem 5.)

Remedy 2. Define trace granularity clearly in the TIM and evaluate trace matrices periodically to ensure that traces are created at the correct granularity. When necessary, you can apply different trace granularities to different subsets of a particular artifact type; however, in this case, the trace matrices should be separated.

Problem 3: Redundant traceability paths. Redundant traceability paths defined in the TIM lead to extraneous and possibly diverging traceability matrices. A TIM includes a redundant path if there's more than one way to trace from one artifact type to another. In some cases, redundant paths might be necessary—for example, it's possible that some requirements are explicitly realized in a UML system design while the remaining ones are directly implemented in the source code. In these cases, use redundant paths judiciously. We observed several cases of redundant traceability paths—for example, the project depicted in Figure 5. If redundant links are stored in different traceability matrices and maintained by different stakeholders, there's a high risk of inconsistency.

In one submission, we observed that test cases traced directly to both hazards and to mitigating requirements. The mitigating requirements then traced to hazards, creating a second indirect trace path from test cases, via mitigating requirements, to hazards. After comparing the trace links along both paths, we found many inconsistencies.

Remedy 3. Minimize and preferably remove redundant traceability paths in the TIM. If required, inform traceability stakeholders about the purpose of each trace route and ensure that each artifact is traced along one path only. Use trace queries to find redundant traces and eliminate them.

Trace Instance Problems

We identified three types of problem at the level of individual trace links. These problems often stemmed from definition problems or from ill-defined day-to-day tracing processes. They affect the reviewer's ability to understand relationships between specific artifacts or to perform a complete coverage analysis of a specific hazard.

Problem 4: Failure to provide unique IDs across the project. Artifacts can lack unique IDs or names. Moreover, IDs might not be used consistently, in which case traceability information exists but is not useful.

A fundamental principle of traceability is that each traceable artifact must have a unique identifier. Furthermore, prefixes used to distinguish artifact types should be unique across the project as well as intuitive to stakeholders. We didn't see this fundamental principle in many of the submissions we observed. For example, requirements identified as SYRS25.01 – SYRS25.xx that trace to tests identified as SYRS01 – SYRSxx are hard to distinguish because the requirements and test all share the single SYRS prefix. This introduces unnecessary communication problems.

In a second example (see Figure 6), one requirement specification consisted of paragraphs containing multiple requirements. IDs were directly embedded in the text and weren't unique across the project. A precise, tool-supported evaluation of the existing traces was almost impossible. Changing existing requirements or adding new ones could lead to inconsistent or illogically ordered labels and section numbers and would require significant effort that would likely introduce labeling mistakes.

A further example illustrates why section headings should not be used in lieu of IDs for tracing purposes. A requirements specification contained a table with alarm and alert definitions. These alarms were validated by separate test cases, giving rise to the need to trace an alarm definition to a test case.

Developers created traces by referring to row numbers provided in the first column of the table. The trace R25.6.2-19 ––> RF-0282 defined in Table 1 refers to row 19 in the table and related it to test RF-0282. Although this scheme is logical, it's also highly vulnerable to changes in the requirements specification. Any changes made to sections, tables, or table rows could break existing traces.

...
During the [...], the timeout SHALL (5.5a) be set to 60 seconds. Upon completion of the [...], the default SHALL (5.5b) be set to 30 seconds.
After the specified interval [...], the [...] SHALL (5.5c) turn off and [...].
...

FIGURE 6. Power-off timeout from our observed examples. The sequential and embedded nature of the IDs makes it difficult to add new requirements without reassigning IDs, resulting in possible synchronization issues among other design artifacts. For trace purposes, IDs must be assigned permanently.

Remedy 4. Artifact IDs are essential for traceability. Carefully define them up front and then consistently use them across all trace links. ID prefixes should allow for an intuitive association with artifact types in a project and should be clearly distinguishable.

Problem 5: Redundant trace information. Duplicated trace information occurs in two different forms. In the first case, identical links are included multiple times in the trace matrix, which will lead to future maintenance problems. In one case we observed, 247 of 2,789 traces were redundant. These 247 traces duplicated 167 unique traces, in some cases, up to six times.

In the second case, a complex form of redundancy occurs when similar traces are established at different levels of granularity. For example, a section in an SRS is traced to a part of the systems' design, but also all requirements in that section are individually traced to the same part. This kind of redundancy is difficult to find, and such links are almost impossible to maintain. We observed multilevel redundant traces in almost all of the documents that we reviewed. The TIM depicted in Figure 5 shows code represented at both the class and method levels, which introduces the possibility of multilevel redundancy problems.

Remedy 5.1. Prevent duplicated links by storing them in a database-like repository. Either define constraints that prevent redundant links from being created or regularly execute trace queries to find duplicated links and remove them.

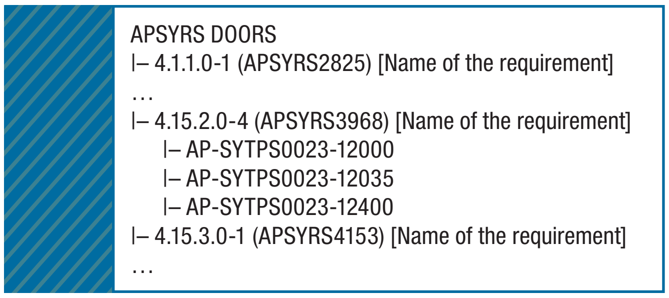
Remedy 5.2. Whenever possible, avoid modeling multiple levels of a single artifact type in the TIM—for instance, model code at either the class or the method level, but not both. When this is unavoidable because different artifact types must be traced to different levels, avoid tracing a single artifact to

TABLE 1

Example from a requirements specification.*

	Condition	Type	Requirements		
...					
19	[...] Canceled	Always on	Pump	MP-6	MP-0
...					

** The table appears in Section 25.6.2 of the specification and provides alarm and alert definitions. Traces refer to the section number followed by the row number in the table. Definitions in rows should carry a unique identifier.*



```
APSYRS DOORS
|– 4.1.1.0-1 (APSYRS2825) [Name of the requirement]
…
|– 4.15.2.0-4 (APSYRS3968) [Name of the requirement]
    |– AP-SYTPS0023-12000
    |– AP-SYTPS0023-12035
    |– AP-SYTPS0023-12400
|– 4.15.3.0-1 (APSYRS4153) [Name of the requirement]
…
```

FIGURE 7. A small excerpt of unprocessed trace information produced by a requirements management tool. This kind of megatable data doesn't provide sufficient information to allow reviewers to directly evaluate product safety.

multiple levels of the same target artifact; that is, trace an individual requirement to either the class level or to the method level, but not to both.

Problem 6: Important links missing. Most certifying or approving organizations expect all hazards to be fully covered through trace links from requirements to code and test cases. Missing links indicate insufficient evidence to evaluate whether a hazard has been fully mitigated. By comparing a project's traces and artifacts, it's possible to identify critical artifacts with no associated links. We found examples of untraced mitigating requirements (widows) and untraced test cases (orphans) in all of the cases we observed. We also observed a case in which an important test was listed as passed, even though it neither appeared with the other tests in the test specification nor in any of the traceability matrices.

Remedy 6. Use trace queries to perform completeness and coverage analysis of project artifacts to ensure that all critical artifacts are traced.

Presentation Problems

We identified three problems in the way traceability data was packaged and presented to FDA reviewers. These problems were pervasive across nearly all of the documents we studied and severely reduced the potential benefits that the traceability information could bring to the assessment process.

Problem 7: TIMs not included in documentation. It isn't sufficient to have a traceability strategy for a project; it's also important to communicate that strategy to all stakeholders, including external assessors. Without the TIM, an assessor or reviewer must invest considerable time to understand the way artifacts are structured and labeled in the project before he or she can start the core assessment task. A TIM provided as part of a project's documentation enables external reviewers to quickly gain an understanding of the development process. (Only one of the documents we observed contained a TIM.)

Remedy 7. Include a TIM in the submitted project documentation so that reviewers can understand artifact naming conventions as well as the traceability paths used throughout the submission.

Problem 8: Traceability links might be presented in megatables. It's relatively easy to generate a megatable of trace links from a database or a requirements management tool, but such tables often fail to include sufficient information about the artifacts, and therefore fail to provide adequate support for claims of product safety. Furthermore, this makes reading and comprehending traces in printed reports almost impossible for the reviewers.

We found several examples of trace matrices spanning multiple columns and tens of pages that included only source and target IDs. Although these traces might be technically correct, their usefulness to someone reading the document is limited. If, for example, a reviewer wants to trace a requirement to related test cases, he or she must find the requirement ID within the (potentially unsorted) megatables, retrieve and remember the related test IDs, and then manually browse the table to find the relevant test cases.

One case presented a barely readable screenshot from IBM's Rational DOORS extending over more than 10 pages, and it didn't provide sufficient information to enable interpretation of the traces (see Figure 7).

Remedy 8. Maintain traces in a table format, but generate useful views, such as trace slices, that support safety inspections. Utilize tracing tools, such as Rational DOORS, that have the ability to generate and print such views.

Problem 9: Traceability as an afterthought. Constructing trace links to merely give the appearance of meeting a regulatory expectation is counterproductive and, if apparent to the reviewer, will diminish confidence in the quality of the development process and the subsequent safety of the delivered system. Furthermore, performing traceability in an ad hoc, after-the-fact fashion means that organizations incur all the costs of creating trace links without experiencing any of its benefits. In several of the submissions we observed, the incompleteness of the trace links and the haphazard effort to document them gave the appearance that traceability had been conducted at the end of the project solely for approval purposes.

Remedy 9. Establish tracing processes and instrument the project environment so that traces are created incrementally and accurately maintained throughout a project's lifetime.

- All traceable artifacts and permitted links are clearly defined in a TIM (Problem 1; Remedies 1.1 and 1.2).
- The granularity of each link is clearly defined. Redundant paths are prevented where possible (Remedies 2, 3, and 5.2).
- All traceable artifacts have been assigned meaningful, unique IDs (Remedy 4).
- Traceability is supported by tools (Problem 2; Remedy 5.1).
- Traces are created throughout the project, rather than after the fact (Problem 3; Remedy 9).
- Traces comply with the TIM (Problem 4; Remedy 5.1).
- Traces are used to perform completeness and mitigation analysis on critical artifacts before submission (Remedies 6 and 9).
- Project documentation and submission contains the TIM, all traced artifacts and all traces (Remedies 7 and 9).
- Traces are reported and submitted as useable views and slices (Problem 5; Remedy 8).
- A project dashboard shows the project state by aggregating trace metrics (Problem 6).

FIGURE 8. A quick traceability checklist.

These practices and remedies highlight the importance of using traceability strategically in safety-critical projects to systematically build a case for product safety and support the assessment process. We're aware that our advice is contrary to some previous papers that advocate a more brute-force approach in which all requirements are thoroughly traced across the life cycle.[3,4] Although we certainly don't discourage complete traceability coverage, we take a pragmatic approach that focuses efforts on creating the trace links needed to support safety analysis. Emerging technologies that use automated methods to dynamically generate just-in-time trace links can be relied on for other less critical tracing needs.

Figure 8 provides a checklist that summarizes some of the main findings of our study and can be used to help a manufacturer to implement good tracing that serves to support product safety claims.

Additional information, tools, and support for performing tracing in safety-critical projects can be found at the Center of Excellence for Software and Systems Traceability at www.coest.org.

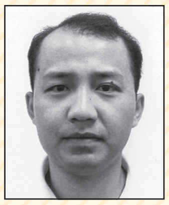
Acknowledgments

The work in this article was partially funded by the US National Science Foundation grant

ABOUT THE AUTHORS

PATRICK MÄDER is a researcher at the Ilmenau Technical University. His research interests include software engineering with a focus on requirements traceability, requirements engineering, and object-oriented analysis and design. Mäder received a PhD in computer science from the Ilmenau Technical University. Contact him at patrick.maeder@tu-ilmenau.de.

PAUL L. JONES is a senior systems/software engineer in the Office of Science and Engineering Laboratories at the US Food and Drug Administration. His research interests include systems and software engineering, safety, and risk management. Jones received an MS in computer engineering from Loyola University. Contact him at paul.jones@fda.hhs.gov.



YI ZHANG is a visiting scientist in the Office of Science and Engineering Laboratories at the US Food and Drug Administration. His research interests include formal methods (especially model-based engineering and software static analysis), software testing, software engineering, and cybersecurity, with an emphasis on introducing research advances in these areas to promote the safety and effectiveness of medical devices. Zhang received a PhD in computer science from North Carolina State University. Contact him at yi.zhang2@fda.hhs.gov.

JANE CLELAND-HUANG is an associate professor in the School of Computing at DePaul University. She also serves as the North American Director of the International Center of Excellence for Software Traceability. Her research interests include the application of machine learning and information retrieval methods to tackle large-scale and safety-critical software engineering problems, especially in the area of software traceability. Cleland-Huang received a PhD in computer science from the University of Illinois at Chicago. She serves on the editorial board for the *Requirements Engineering Journal* and *IEEE Software* and as associate editor for *IEEE Transactions on Software Engineering*. Contact her at jhuang@cs.depaul.edu.

#CCF-0810924, the Austrian Science Fund (FWF): M1268-N23, and the German Research Foundation (DFG): Ph49/8-1.

The FDA does not endorse any tools or techniques mentioned in this article.

References

1. P. Bishop and R. Bloomfield, "A Methodology for Safety Case Development," *Proc. 6th Safety-Critical Systems Symp.*, F. Redmill and T. Anderson, eds., Springer, 1998, pp. 194–203.
2. J. Cleland-Huang et al., "Trace Queries for Safety Requirements in High Assurance Systems," *Proc. 18th Int'l Conf. Requirements Eng.: Foundation for Software Quality* (REFSQ 12), Springer, 2012, pp. 179–193.
3. O. Gotel and C. Finkelstein, "An Analysis of the Requirements Traceability Problem," *Proc. 1st Int'l Conf. Requirements Eng.*, IEEE CS, 1994, pp. 94–101.
4. B. Ramesh and M. Jarke, "Toward Reference Models of Requirements Traceability," *IEEE Trans. Software Engineering*, vol. 27, no. 1, 2001, pp. 58–93.
5. J. Cleland-Huang, O. Gotel, and A. Zisman, "Software and Systems Traceability," Springer, 2011; doi:10.1007/978-1-4471-2239-5.
6. P. Mäder, O. Gotel, and I. Philippow, "Motivation Matters in the Traceability Trenches," *Proc. 17th Int'l Conf. Requirements Eng.* (RE 09), IEEE CS, 2009, pp. 143–148.
7. P. Mäder and J. Cleland-Huang, "A Visual Language for Modeling and Executing Traceability Queries," *J. Software and Systems Modeling*, Apr. 2012; doi:10.1007/s10270-012-0237-0.
8. *ANSI/AAMI/IEC 62304:2006, Medical Device Software—Software Life Cycle Processes*, Assoc. Advancement Medical Instrumentation, 2006.
9. A. Frisoli et al., "Arm Rehabilitation with a Robotic Exoskeleleton in Virtual Reality," *Proc. IEEE 10th Int'l Conf. Rehabilitation Robotics* (ICORR 07), IEEE, 2007, pp. 631–642.
10. J. Cleland-Huang et al., "Best Practices for Automated Traceability," *Computer*, vol. 40, no. 6, 2007, pp. 27–35.