Published in Proceedings of the the 36th International Conference on Software Engineering (ICSE), Hyderabad, India
@INPROCEEDINGS{conf/icse/ClelandHunagGHMZ14, author = {Cleland-Huang, Jane and Gotel, Olly and Hayes, Jane Huffman and M{\"a}der, Patrick and Zisman, Andrea}, title = {Software Traceability: Trends and Future Directions}, booktitle = {Proc. of the 36th International Conference on Software Engineering {(ICSE)}, Hyderabad, India}, year = {2014}}

Software Traceability: Trends and Future Directions

Jane Cleland-Huang
DePaul University
SAREC
Chicago, IL, USA
jhuang@cs.depaul.edu

Olly Gotel
Independent Researcher
New York, NY, USA
olly@gotel.net

Jane Huffman Hayes
Department of Computing
University of Kentucky
Lexington, KY, USA
hayes@cs.uky.edu

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

Andrea Zisman
The Open University
Department of Computing
Milton Keyes, UK
andrea.zisman@open.ac.uk

ABSTRACT

Software traceability is a sought-after, yet often elusive quality in software-intensive systems. Required in safety-critical systems by many certifying bodies, such as the USA Federal Aviation Authority, software traceability is an essential element of the software development process. In practice, traceability is often conducted in an ad-hoc, after-the-fact manner and, therefore, its benefits are not always fully realized. Over the past decade, researchers have focused on specific areas of the traceability problem, developing more sophisticated tooling, promoting strategic planning, applying information retrieval techniques capable of semi-automating the trace creation and maintenance process, developing new trace query languages and visualization techniques that use trace links, and applying traceability in specific domains such as Model Driven Development, product line systems, and agile project environments. In this paper, we build upon a prior body of work to highlight the state-of-the-art in software traceability, and to present compelling areas of research that need to be addressed.

Categories and Subject Descriptors

D.2.1 [**Software Engineering**]: Requirements/Specifications

General Terms

Documentation

Keywords

Software traceability; road map

1. INTRODUCTION

Software traceability has long been recognized as an important quality of a well-engineered software system [37, 78]. Defined by the Center of Excellence for Software and Systems 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" [14], traceability is a required component of the approval and certification process in most safety-critical systems. For example, the DO-178C standard [73], which the USA Federal Aviation Administration (FAA) has established as the means of certifying that software aspects of airborne systems comply with airworthiness requirements, specifies a very detailed set of traceability requirements including the need to provide "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 [30].

Despite its importance, traceability is perhaps one of the most illusive qualities of the software development process. The cost, effort, and discipline needed to create and maintain trace links in a rapidly evolving software system can be extremely high. Moreover, its benefits often go unrealized in practice, either due to ill-defined and ad-hoc traceability processes, poor user training, or a lack of effective tooling [37, 5]. Reflecting this state of practice, the "Critical Code: Software Producibility for Defense" report commissioned by the U.S. Department of Defense identified requirements traceability as one of the seven technology areas on which research should be targeted in order to assure the safe and correct operation of current and future software intensive systems [15].

In this paper, we set out a focused research agenda for software traceability. We build upon previous work that first identified the Grand Challenge of Traceability [35] and then proposed a high-level roadmap in order to achieve it [34]. Motivated by current practice, this paper draws out and drills down on the key areas in which research focus is needed. It details the state of the art in each of these areas and then focuses attention on specific high priority research needs. In Section 2, we present three different per-

ICSE '14 Hyderabad, India

spectives which are used to discuss traceability problems and to present a research agenda. Sections 3 to 5 use this framework to explain seven broad research areas, and to outline specific research needed in each of these areas. Section 6 concludes the paper.

2. CURRENT PRACTICE

To assess the current state of the practice, we reviewed recent literature describing industrial studies of traceability [63, 75, 71, 9]. We also asked eight seasoned practitioners from the healthcare, enterprise information systems, military, and transportation sectors for their perspectives on traceability practice and challenges. One of the interesting, if not entirely surprising, findings is that knowledge and acceptance of traceability tends to be realized on a spectrum, from fast-paced agile-like projects and/or business-oriented applications on one end, to slower-paced, carefully planned, safety-critical projects on the other.

On the less formal end of the spectrum, the vast majority of practitioners do not even understand the "traceability" term. For example, five out of eight IT project managers, with experience on varying sized banking and administrative projects at an IT company in the Netherlands, had no understanding of the concept [8]. There are also non-believers, i.e. practitioners who understand the idea of traceability, but simply see it as an unnecessary evil. For example, the owner of one rapidly expanding IT company in the UK referred to traceability as a "made up problem."

In contrast, practitioners working in regulated and/or safety critical domains on the more formal end of the spectrum, where traceability is governed by standards, have a far more comprehensive understanding of what traceability does and how it should be established. Unfortunately, that does not mean that they implement it more willingly or more effectively. A recent analysis of the traceability documents submitted to the FDA as part of the medical device approval process revealed numerous problems related to the overall completeness and correctness of the trace data [64]. Not only was traceability data incomplete, incorrect, and conflicting in many cases, there were clear indications that trace links had been created at the very end of the process in many projects, specifically for certification processes. This conjecture was confirmed by a serendipitous interview conducted with an employee from a medical device company, who stated that it was common practice for all traceability data to be created and documented immediately prior to certification.

There are however, several examples of traceability being achieved effectively and as intended in practice. Panis reports how Teradyne successfully integrates tracing into their development process using the Siemens TeamCenter tool [75]. His survey showed that 20% of systems engineers, 70% of subsystems engineers, and 80% of design engineers reported that their traceability efforts were worthwhile. They listed the benefits as reduced effort during change management, coverage analysis (i.e. assessing missing requirements), and CMMI certification achievements. Similarly, Nistala described Tata Consulting Services' successful use of traceability across various phases of the life-cycle [71], and several of the practitioners we talked with described somewhat effective, albeit rather narrow, adoption of traceability to support specific tasks such as testing or regulatory compliance. Synthesizing this background information suggests that traceability is successfully implemented in some projects within some organizations while the majority of projects fail to achieve effective traceability or incur excessive costs in so doing.

3. FUTURE VISION

Researchers and practitioners from the CoEST have worked together since 2005 to establish a vision and subsequent roadmap for advancing the state of practice in software traceability [35, 34]. Driven by clearly articulated research challenges, this vision seeks to achieve *ubiquitous* traceability that is "always there" and "built into the engineering process." In this vision, the cost and effort of establishing and maintaining traceability basically disappears as trace links are generated automatically by tools as a byproduct of the development process. Benefits are realized across all projects, not just regulated ones. Traceability information is made accessible to humans to support the tasks that are relevant to their project environments, and rendered in ways that facilitate interaction and decision-making. The project environment self-adapts as a project evolves and also learns from human feedback. Ubiquitous traceability is achieved automatically, as a result of collecting, analyzing, and processing every piece of evidence from which trace data can be inferred and managed.

For example, a new developer joins an agile team and is assigned a user story to implement. She uses automatically captured trace information to explore the impact of the new story on the system. Results are quickly visualized in ways that help her to understand which parts of the codebase might need to be changed, potential side effects on existing user stories and test cases, and a list of fellow team members who have previously worked with the code and could be considered expert consultants. Similarly, in a safety-critical system, team-members preparing for certification request a report that returns trace slices for each identified hazard, showing its contributing faults, related mitigating requirements, associated code, test cases and logs, and also supporting rationales explaining how these artifacts mitigate the hazard in the deployed system. Achieving the vision will demand many years of research in a number of areas.

We structure our presentation of future research needs around the three perspectives of *goals*, *process*, and *technical infrastructure*, each of which has a unique impact upon traceability. The goal-oriented perspective is inherently visionary, while the other perspectives address the contexts in which traceability must be deployed. The need for different perspectives highlights the complexity of the traceability problem and the naïvete of approaching traceability research from a single angle, such as a purely algorithmic one. Solving traceability problems requires a cross-disciplinary approach that integrates novel algorithmic solutions with enterprise and project level planning, process improvement initiatives, tooling, systems engineering solutions, and human factors.

In the following sub-sections, we briefly describe each perspective and examine the distribution of research effort across these areas in the past decade. Insights gained from this analysis provide a compelling argument for expanding and redistributing research efforts.

3.1 A Goal-Oriented Perspective

An earlier roadmap [34], produced by researchers in the traceability community, identified a number of challenges

Table 1: A Goal-Oriented Perspective

Goal 1: Purposed	**Traceability is fit-for-purpose and supports stakeholder needs (i.e., traceability is requirements-driven).** Prototypical stakeholder requirements for traceability need to be clearly defined and measurable for specific software and systems engineering tasks.
Goal 2: Cost-effective	**The return on investment (ROI) from using traceability is adequate in relation to the outlay of establishing it.** Develop techniques for computing the ROI of traceability in a project, understanding the impact of various traceability decisions at various stages of the life-cycle upon both the cost and benefits of the traceability process.
Goal 3: Configurable	**Traceability is established as specified, moment-to-moment, and accommodates changing stakeholder needs.** Develop techniques for dynamically generating and maintaining accurate and semantically rich trace links that are configured according to the current needs of the project.
Goal 4: Trusted	**All stakeholders have full confidence in the traceability, as it is created and maintained in the face of inconsistency, omissions and change; all stakeholders can and do depend upon the traceability provided.** Develop techniques for assessing and communicating the current state of traceability in a project, and develop self-adapting techniques so that quality is preserved in the face of change.
Goal 5: Scalable	**Varying types of artifact can be traced, at variable levels of granularity and in quantity, as the traceability extends through-life and across organizational and business boundaries.** Develop techniques for scaling up traceability techniques, and for supporting multi-grained traceability across a variety of artifact types and organizational boundaries.
Goal 6: Portable	**Traceability is exchanged, merged and reused across projects, organizations, domains, product lines and supporting tools.** Develop policies, standards, and formats for exchanging and integrating traceability information across projects and organizations.
Goal 7: Valued	**Traceability is a strategic priority valued by all; every stakeholder has a role to play and actively discharges his or her responsibilities.** Develop supporting techniques that cross the technical and business domains of a project so that the benefits of traceability are visible and accessible to all stakeholders.
Grand Challenge: Ubiquitous	**Traceability is always there, without ever having to think about getting it there, as it is built into the engineering process; traceability has effectively "disappeared without a trace."** Achieved only when traceability is established and sustained with near zero effort.

for traceability, including the *Grand Challenge* to achieve Ubiquitous Traceability. These challenges are reproduced as goals in Table 1. Each goal represents a desired *quality* of

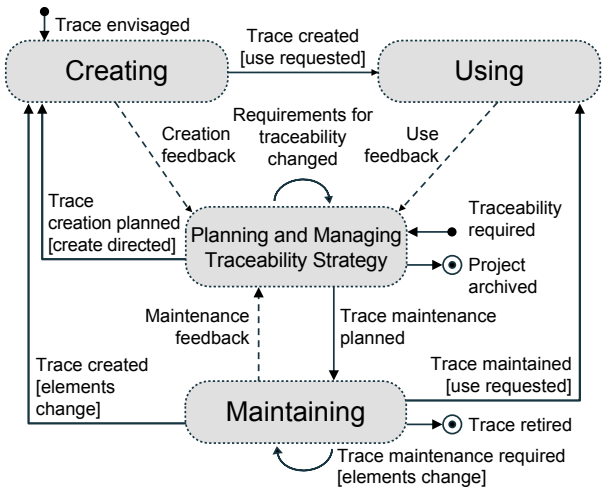


Figure 1: The Process Perspective

traceability, and is refined into a set of research topics and practices (described in detail in [35]). For example, research topics to achieve the *purposed* goal include (Purposed.RT1) developing prototypical stakeholder requirements for traceability, including scenarios of use, and (Purposed.RT6) performing empirical studies to determine whether traceability is fit for purpose. In other words, to achieve purposed traceability, we need to better understand the tasks that traceability is needed for, and evaluate how well any instance of traceability supports them. Only then can we develop appropriate techniques, tools and processes to put the required traceability in place to support these tasks. Additional focus is on the delivery of *cost-effective*, *configurable*, *trusted*, *scalable*, *portable*, and *valued* traceability.

3.2 A Process-Oriented Perspective

The process-oriented perspective, depicted in Figure 1, focuses upon the primary areas associated with the traceability life-cycle: *planning and managing a traceability strategy*, and *creating*, *maintaining*, and *using* traceability [36]. At the start of a project, the traceability strategy is planned, and the project environment is instrumented accordingly. The project's traceability is then implemented and managed as the project proceeds. Trace links are created, used, and maintained according to the strategic plan.

3.3 A Technical-Infrastructure Perspective

The significant practical challenges associated with instrumenting a project environment for traceability are highlighted by this perspective. A tracing context must minimally include functions for storing and retrieving physical data; supporting strategic planning; physically creating and maintaining trace links; executing trace queries; and interacting with the trace user. This is depicted in Figure 2. Complexities are introduced by the fact that trace links must be created and maintained across heterogeneous artifacts (e.g. requirements, code, and test cases), that may reside in a variety of third party tools, and often across organizational boundaries. Infrastructure and supporting algorithms must therefore be developed to support cross-organizational traceability and data integration across a wide variety of tools and data formats. If such issues are not addressed, it

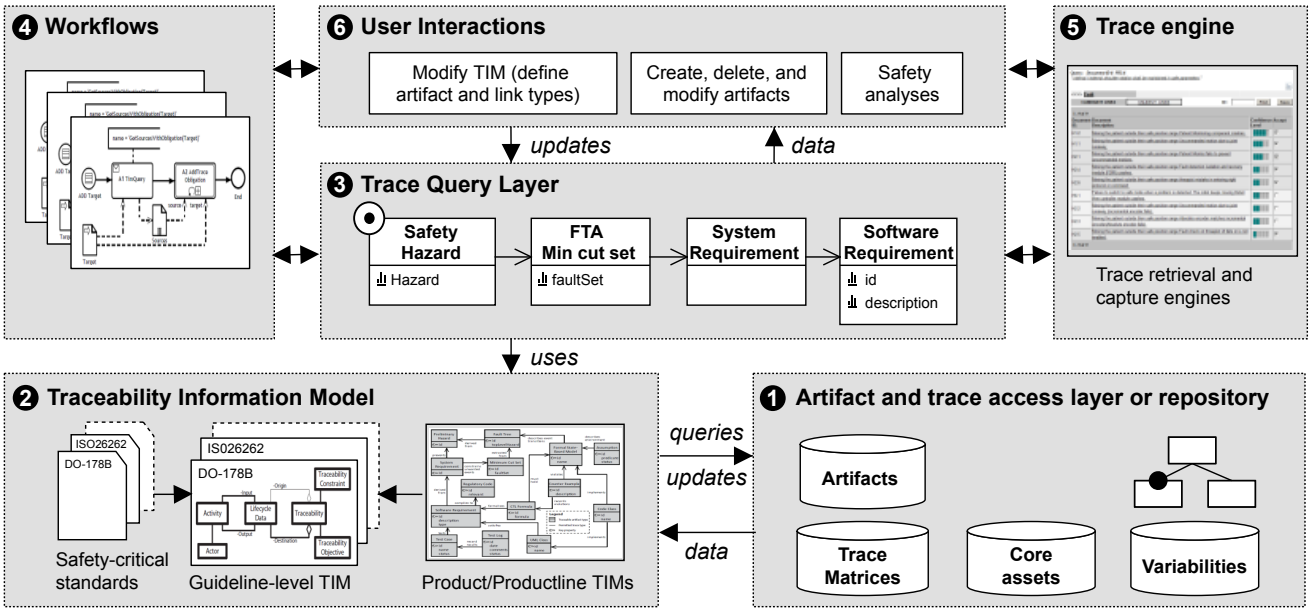


Figure 2: The Technical Perspective

is unlikely that advances in traceability research will be able to make a significant impact on industrial practice.

3.4 Integrating Perspectives

These three perspectives are interdependent. Take the *scalable* goal as an example. A scalable tracing solution seeks levels of abstraction and granularity in traceability techniques, methods and tools, facilitated by improved trace visualizations, to handle large datasets and the longevity of these data [34]. At the process level, this means solutions that help stakeholders to determine the right level of granularity and to facilitate tracing at that level. In large and/or complex systems, with potentially hundreds of thousands of trace links, algorithms need to be fine-tuned to return accurate and timely results, and visualizations need to be scaled to present information in consumable pieces. The infrastructure needed to support this functionality needs to be embedded across multiple tools and environments. An emphasis on how the stakeholder interacts with and handles issues of scale is a given.

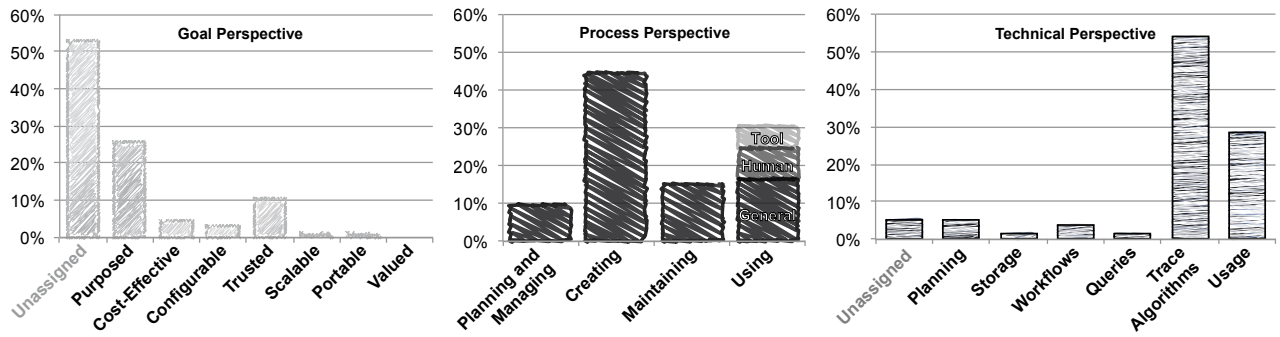
A DBLP search of selected publications from 2003-2013 (IEEE Transactions on Software Engineering, ASE, ICSE, RE, ASE, Software, and ICSM) was carried out using the search term *Trace*. Filtering out papers not directly related to software traceability returned 72 papers. Each paper was then classified according to elements of the goal, process, and technical perspectives and results are reported in Figure 3.

From the goal-oriented perspective, over 50% of the papers were only mappable to the overarching goal of "ubiquitous" traceability and could not be easily classified against any of the supporting ones specifically. 26% of the papers were classified as *purposed*, primarily because they addressed specific uses of traceability. However, there were no papers which directly addressed the meta-problem of purposeful traceability. 11% of papers addressed problems of human evaluation and were mapped to the *Trusted* category. All other goals had 5% or less of the associated papers.

From the process perspective, 10% of papers mapped to *planning* topics, 44% mapped to *creating*, 15% to *maintaining*, and 31% to *usage*. Of the *usage* related papers, 53% focused on using traceability to support general SE tasks, 30% focused on human interactions, and 17% on tool support. Similarly, in the technical perspective the lion's share of papers, i.e. 54%, were associated with trace algorithms, 28% with user interactions, and 5% or less with each of the other areas.

Despite these numbers it would be a mistake to correlate research effort with criticality (or priority) of the addressed problem. Researchers may choose to focus on specific problems for a number of reasons including (1) importance of the problem, (2) availability of data, (3) interests of the research group, and (4) perceived ability to publish results. Factors 2-4 all serve to inhibit researchers from addressing certain types of pressing research challenges. For example, researchers have demonstrated a propensity to favor the *trace creation* process area as it is intellectually stimulating, datasets are readily available, and research questions can be easily formulated and experimentally evaluated. In contrast other equally critical research problems, such as those related to processes for traceability strategizing, tend to attract less research attention.

Furthermore, the research emphasis only partially aligns with the concerns of industrial practitioners. In a survey of 56 traceability-applying practitioners [9], participants reported that their primary uses of traceability centered around requirements engineering and management, project management, and compliance demonstration and that they were less likely to use traceability to support design, implementation, and maintenance activities. Additional feedback from survey respondents also suggested that the lack of customizable tools hindered their use of traceability. This suggests that the academic interest in algorithm development needs to be more clearly contextualized to support the management and compliance concerns of practitioners, and that we need to continue efforts to support technology transfer through developing prototypical industry-ready tools.



(a) Goal perspective (b) Process perspective (c) Technical perspective

Figure 3: Traceability Research by Perspective over the past decade (2003-2013)

3.5 Research Focus Areas

Based on these findings, we set out to identify compelling areas of traceability need. As a result, we have identified 7 specific areas for research in this paper. We describe these in the context of the process-oriented perspective and make connections to the other two perspectives.

Under the umbrella of traceability planning and management (Section 3), we focus upon (1) understanding stakeholder needs, driven by their roles and tasks, and (2) developing techniques to support traceability strategizing, so that we can deliver useful, value-filled tracing solutions that are customized to the specific needs of a project. Under the umbrella of creating and maintaining traces (Section 4), we explore (3) trace creation, (4) trace maintenance, and (5) trace integrity, so that accurate and trusted trace links can be created and maintained throughout a project. Under the umbrella of using traces (Section 5), we discuss (6) accessing and querying trace data, and (7) making sense of trace results, through visualization.

The approach taken in this paper builds upon our earlier road-mapping initiative [34], but differs in two important ways. First, in the structure: it uses the traceability process as a framework to highlight current research trends and future directions. Second, in the content: it prioritizes concrete research that needs to be undertaken to support advances in the process areas. The resulting research directions form an actionable research agenda. They are mapped back to research topics identified in the Grand Challenge of Traceability report where appropriate [35]. Taking this alternative approach has not only highlighted which areas of the earlier roadmap are the most significant to work toward, it has also pointed to a number of research gaps.

Figure 4 maps each of the research directions onto their relevant process and technical areas. It also highlights the quality goals they address. For some of these goals we provide explicit mappings throughout the remainder of the paper, while others represent more general relationships and are shown only in the diagram.

4. PLANNING AND MANAGING

There is no "one-size-fits-all" approach to software traceability. Projects of different types, sizes, and criticality levels all have their own reasons for requiring traceability and their own unique constraints that influence its implementation. Traceability planning and managing involves determining the stakeholder and system requirements for traceability on a project, and designing a suitable traceability strategy to enable them to be satisfied [36]. These are the first two focal areas in which research is needed.

4.1 Understanding Stakeholder Needs

Traceability research must be driven by the needs of its stakeholders, who ultimately adopt tracing solutions to support activities such as requirements satisfaction assessment, impact analysis, compliance verification, or testing effort estimation. These tasks are performed by a variety of stakeholders including requirements analysts, safety analysts, certifiers, reverse engineers, developers, architects, maintainers, and Verification and Validation (V&V) analysts (possibly Independent). Unfortunately, there is little prior work that examines the specific needs of the stakeholder in the traceability process and, as a result, academic researchers are left making assumptions about industry needs.

4.1.1 Motivating Example

An IV&V analyst or an external certifier for a safety-critical system must ensure that the right system has been built (validated) using the proper processes (verification). At each phase of the life-cycle, the current artifact must be examined to ensure that it "satisfies" the artifact from the prior life-cycle phase. For example, the requirements specification, originating from the requirements phase, must be fully satisfied by the design specification(s) developed during the design phase. A stakeholder is responsible for creating a mapping from design elements to requirements and documenting it in a requirements traceability matrix (RTM).

To perform satisfaction assessment, the IV&V analyst needs to consider the requirement elements at a fine-grained level. For example, a requirement R001 *The system shall enable users to process 500 connections at any given time. Each connection may have up to 10 users.* will be broken down into two sub-requirements such as *R001a: The system shall enable users to process five hundred connections at any given time*, and *R001b: Each connection may have up to 10 users.* The IV&V analyst must then read each sub-requirement, and decide if it has been satisfied, partially satisfied, or not satisfied by the collection of design elements that are mapped to it within the RTM. The analyst may use a number of heuristics to perform this work, such as looking for synonyms in phrases, or looking up the meaning of acronyms used in phrases. This is a time consuming, error prone task that requires examining the semantics of each element. Even if the tasks are semi-automated, the results must be vetted

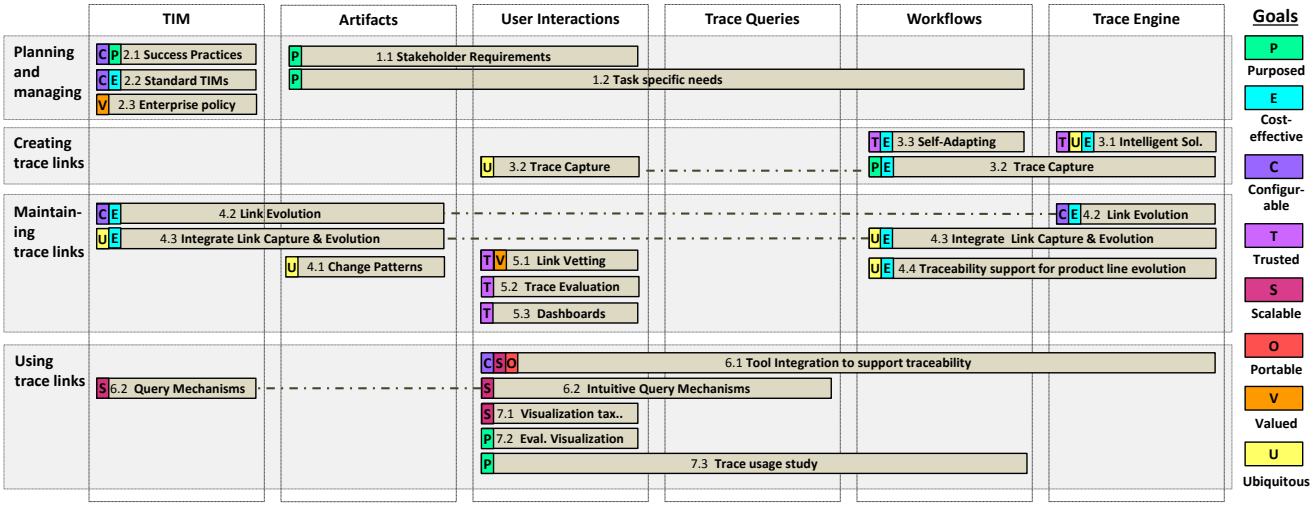


Figure 4: Research Directions mapped to the Three Perspectives

by an IV&V analyst, as only the human analyst can make the final decision on whether a requirement has been fully satisfied. As depicted in Table 2, each of these tasks can be supported, at least partially, by automation.

4.1.2 *Task-Driven Traceability*

Unfortunately, the traceability community has not invested much effort to understand tracing tasks at the level of detail shown in this example. Some exceptions include the work by Ramesh and Jarke [78] who conducted an extensive industrial study and developed several traceability reference models that captured artifact types and the links needed to support specific tasks such as requirements to design allocation. However, their study did not investigate how these links were used to support higher level software engineering tasks. Other researchers have investigated the use of traceability within specific contexts. For example, Von Knethen [89] and Conte de Leon [18] both described techniques for using trace links to support impact analysis, while Mäder et al. [60] and Poshyvanyk et al. [77] explored its use for supporting software maintenance and for bug fixing [60].

4.1.3 *Research Directions*

This area of research is driven by the goal for traceability to be *purposed*. From the process perspective, *planning and managing* is at the heart of and influences all other areas of the traceability life-cycle. It impacts the *user interactions* layer of the technical infrastructure. We believe the greatest gains in the area of traceability planning and management can be achieved if we focus efforts on the following research. (Note that we provide a cross reference (in italics) back to research topics listed in the Grand Challenge of Traceability report where appropriate [35].)

- **RD-1.1 Develop prototypical stakeholder requirements** for traceability, including scenarios of use *(Purposed. RT1)*.
- **RD-1.2 Empirically validate task-specific traceability techniques** as applied by stakeholders *(Purposed.RT6)*.

4.2 Traceability Strategizing

Without an upfront tracing strategy, projects tend to produce ad-hoc, inconsistent, incomplete, conflicting trace links, even in many safety-critical projects [64]. Furthermore, given the cost-benefit debate surrounding traceability [5, 46], the natural tendency is to put only absolute essential traceability in place to address the immediate concerns and visible needs of a project. For example, as previously mentioned, traceability may be established immediately prior to the external certification or approval process in a safety-critical project, instead of systematically throughout the entire development process.

4.2.1 *Traceability Guidelines*

While practitioner-oriented bodies of knowledge, handbooks, and process improvement frameworks describe the need for traceability in general terms (e.g. the BABOK, CMMI, INCOSE Systems Engineering Handbook, IREB, PMBOK, SPICE, SWEBOK, etc.), and while international standards routinely demand traceability (e.g. IEEE-STD-830-1998, IEEE-STD-1220-2005, IEEE-STD-15288-2008, ISO-29148-2011, etc.), explicit guidance and assessment on particular practices for setting up traceability is scarce.

The landscape is better in the safety-critical domain, where standards such as the FAA's DO-178c [29] provide detailed traceability guidelines for various levels of system criticality. For example, DO-178c explicitly states that in high-criticality systems, traceability must be provided between "source code and low-level requirements" in order to verify "the absence of undocumented source code." Such guidelines often specify traceability paths (i.e. that trace links must be established between particular types of source artifacts and target artifacts), and they often also specify the purpose of establishing the link [80]. What they fail to do is to specify the granularity of the links and the optimal trace path (i.e. the path through the graph from source to target artifacts). For example, at the source code level, it is unclear whether trace links should be established at the method, class, file, or module level, and the benefits of specific trace paths (i.e. linking source code directly to requirements versus linking to requirements via design) are also not well understood or documented. Furthermore, we have observed that in practice, requirements, even for safety-critical systems, are often not primitive, not unambiguous, not assigned unique IDs, etc., and therefore are often difficult to trace [64].

Table 2: Requirements Satisfaction Tasks

Task	Performed using
Issue a (reusable) trace query to perform Satisfaction Assessment	Trace queries e.g., SQL, VTML, TQL
Build RTM between artifact pair	Trace engine e.g., using LSA, VSM, plus supporting features
Browse/Search/Eval. links	Filterer/Visualizer
Determine level of satisfaction	Decision support tool (uses thesaurus, heuristics etc.)
Visualize satisfaction results	Visualization alg/tools
Provide feedback on results	User

4.2.2 Reference Models

To date, the most significant and almost exclusive contribution of the research community to traceability strategizing has been on reference traceability for standard projects. A traceability reference model specifies the permissible artifact types and permissible link types that can form a trace on a project, and is derived from an analysis of the queries that the resulting traceability is intended to answer [78]. Despite several proposed traceability reference models, and high interest in them from the research community, there is none that is universally accepted or widely used in industry, neither generic nor domain-specific. An exception to this could become the work of Katta [48], which defines a detailed reference model for use in the highly-regulated nuclear domain. The model consists of over 100 different artifact types and relating link types needed to support the construction of a safety case. The fact that the definition of the model was driven by the industry itself could benefit its acceptance.

4.2.3 Traceability Information Models

More recently, attention has been directed toward defining and using project-specific meta-models for traceability strategizing [62]. The resulting traceability information model (TIM) is effectively an abstract expression of the intended traceability for a project [36]. Work on how TIMs can be designed, adapted and employed as part of an end-to-end traceability process is now emerging. This focus of the research community has undoubtedly centered attention on the goal-directed and query-oriented nature of traceability (See 5.1.2), contributed to a deeper understanding of the semantics of trace artifacts and links [84], and made the planning of rich and intelligent forms of traceability feasible [45].

However, this is insufficient. Important value decisions need to drive discussions about trace artifacts, links, and capture mechanisms. Parallel research has therefore focused on those issues that frame the initial planning and management of traceability. Promising examples include a value-based approach to determine what traces to capture, and adaptive strategies to effectively migrate between simple and more detailed forms of traceability as needs evolve [26]. Case study reports provide mature insights into industrial practices for planning and managing traceability [51, 79, 78, 75, 71, 38], suggesting that the potential for researchers and practitioners to work together to synthesize lessons and inform practice is now mature.

Trace strategies in the form of reusable TIMs, templates, economic models, and industry guidance, can be reused across projects. From the infrastructure perspective, all of the layers shown in Figure 2, including the TIM, Trace Query Layer, Workflows, Trace engine, and User interactions are technically reusable [12], although there are no published studies on their reuse in industry.

4.2.4 Strategy Customization

Because systems evolve over time, and because perfect traceability cannot be one hundred percent planned for upfront, self-managing traceability systems will need to configure, grow and repair organically to address changing contexts and needs. Traceability strategizing will be intrinsic to project management practices, facilitated by intelligent toolkits that help to devise just the right traceability strategy based on an assessment of evolving needs and available resources. Strategies will be designed dynamically to meet the traceability needs and economic pressures of any particular project's context.

Our current ideas of traceability are going to be stretched by demands to scale in many dimensions, with constant demands to improve reliability and performance, all in the absence of perfect trace data. Traceability will have to be planned for and managed in significantly new ways to account for uncertain environments and partners, different development and documentation practices, and to extend beyond the realm of software assets.

4.2.5 Research Directions

Traceability strategizing is primarily related to the *Cost-effective* goal which states that the "The return on investment from using traceability is adequate in relation to the outlay of establishing it." It anchors planning and management processes and is supported primarily in the *Traceability Information Model* layer of the technical infrastructure. We define important research in the area as follows.

• **RD-2.1 Identify ingredients for through-life traceability success** in different contexts, from a thorough understanding of industry best and worst practice, and then use this knowledge to establish a process framework to guide practitioners, develop standards, inform tools, and enable training. *(Purposed.{RT4,RT7}; Configurable.RT1)*

• **RD-2.2 Prepare a family of standardized TIMs** and usage guidance. Adaptable and extensible meta-models need to provide the capability for a project or organization to grow its traceability competence via well-defined paths. *(Configurable.RT3)*

• **RD-2.3 Develop policies and protocols** that enable the traceability of any desired corporate asset to be planned for and established across the enterprise. These clarify how the data is to be exchanged and trusted, but permit autonomy of implementation. *(Portable.RT3)*

5. CREATING & MAINTAINING TRACES

The Grand Challenge of Traceability report [35] lists one of the research tasks for achieving ubiquitous traceability as "total automation of high-quality trace creation and maintenance" *(Ubiquitous.RT2)*. The quest to achieve this goal has attracted significant levels of interest in the research commu-

nity. As previously discussed, over 50% of the traceability-related research papers that we evaluated have focused on the automated creation and/or maintenance of trace links. In this section, we describe the three essential research areas of (3) trace creation, (4) trace maintenance, and (5) trace integrity, all of which must work synergistically in order to automate the trace creation process and work toward the goal of completely eradicating manual traceabilty effort. Realistically, the state-of-the art provides only semi-automated support for tracing and it is unlikely (and possibly undesirable) for the human to be entirely eliminated from the loop. We therefore also point to the research needed to facilitate and integrate the human seamlessly.

5.1 Trace Creation

Current work in the area of trace creation primarily falls under the two distinct categories of (1) trace retrieval, and (2) prospective trace capture. We discuss each of these.

5.1.1 Trace Retrieval

The idea behind trace retrieval is to dynamically generate trace links between *source* and *target* artifacts. For example, to retrieve all relevant Java classes for a given requirement, or retrieve all requirements that demonstrate compliance to a specific regulatory code.

The current approaches build upon seminal work of Antoniol et al. [3], who used a probabilistic approach to retrieve trace links between code and documentation. The underlying concept was that information retrieval (IR) algorithms could be used to estimate the similarity between two documents. Building on this concept of developing a corpus of all terms in an artifact pair, and then representing each element based on these terms, researchers in the software maintenance, requirements engineering, and reverse engineering communities have examined numerous algorithms and their combinations. For example, Hayes et al. used the Vector Space Model (VSM) IR algorithm in conjunction with a thesaurus to esablish trace links [44]. Results from this, and other related studies, showed that while the majority of relevant artifacts were correctly retrieved, the techniques also returned too many false positives. This lead to an emphasis on precision, with many ensuing ideas on how to increase it. Latent Semantic Indexing [19] and Latent Dirichlet Allocation (LDA) [22, 6] were applied in an attempt to understand the semantics context of elements in the artifacts, thus leading to higher quality links. Some researchers examined the idea of enhancing accuracy by merging results from individual algorithms [22, 32]. Less traditional approaches to improved trace quality have included AI swarm techniques [85] and techniques which combine heuristic rules with trace retrieval techniques [84, 41].

Unfortunately, improvements results seem to have plateaued with new techniques returning only minor increases in recall and precision. The problem is primarily caused by term mismatches across documents to be traced. Looking ahead, we therefore need to explore alternate trace creation techniques that circumvent limitations of term mismatches. Three promising approaches include incorporation of (1) runtime trace information, (2) general and domain-specific ontologies, and (3) special case strategies. A fourth area related to user feedback is discussed in Section 5.3.1.

In the closely related field of feature location, researchers have explored the **integration of runtime traces** with static trace recovery techniques [25]. This approach comes with the price of instrumenting the runtime environment, and is limited to tracing artifacts such as code which can be executed either actually or symbolically.

Ontology and project glossaries have also been used to address the term-mismatch problem by connecting related concepts. Several researchers have explored this notion [41, 43] and have developed various approaches for using concepts in the ontology to connect otherwise dissimilar terms. Humans performing tracing tasks implicitly make these connections. Future automated trace retrieval techniques need to similarly understand the concepts of the domain and be able to connect terms via a mental concept map.

One other interesting direction borrows a page from industrial search engines, such as Google, which include numerous **special case algorithms**. For tracing purposes, if researchers can identify hundreds, or even thousands, of special cases, and can apply them systematically and reliably, then when combined together, we can expect significant improvements in the overall quality of generated trace links.

5.1.2 Prospective Trace Capture

Another orthogonal approach to trace creation sets out to capture and infer trace links from the project environment and from the actions of the project stakeholders. This approach is appealing because trace links can be inferred as a natural byproduct of software engineering activities. Work in this area is divided between techniques which instrument the general project environment to monitor the actions of developers [6] and those which infer trace links from tagged items in the logs of version control systems and other kinds of repositories [40, 23]. Asuncion et al. pioneered the work in prospective trace capture through developing plugins to instrument a project environment and track artifacts accessed in close succession by project stakeholders. From this information, they were able to infer trace links [6].

While prospective link capture is appealing, the challenge of instrumenting an environment by building and evolving plugins for many different versions of different tools, distributed across potentially different networks, is quite challenging. Such approaches require researchers to address systems and software engineering issues that go far beyond algorithmic solutions and reach into the technical infrastructure of the project, particularly into the *user interaction layer* where plugins reside, the *repository* layer where data are stored, and the *trace engine* layer which needs to implement algorithms that make sense of the captured data.

5.1.3 Self Adaptation

Self-aware systems are able to modify their own behavior in an attempt to optimize performance. Such systems can self-diagnose, self-repair, adapt, add or remove software components dynamically, etc. [88]. Initial work has investigated adaptation in traceability environments. For example, Poshyvanyk et al. used a Genetic Algorithm (GA) to discover the best way to parameterize Latent Semantic Indexing (LSI) [74]. Falessi et al. [28] used regression analysis to discover the right combination of techniques for a given dataset. Finally, Lohar et al. [58] modeled available features of the trace engine (e.g. stoppers, stemmers, VSM, LSI, LDA, voters, etc.) in a feature model, and used a genetic algorithm to search for the ideal configuration. They showed, on several large industrial datasets, that incremen-

tal customization can lead to significant improvements in the quality of generated trace links, sometimes as much as 100% over a more standard VSM configuration.

5.1.4 Research Directions

To work toward achieving automated trace creation, we propose the following priority research.

• **RD-3.1 Develop intelligent tracing solutions** which are not constrained by the terms in source and target artifacts, but which understand domain-specific concepts, and can reason intelligently about relationships between artifacts. *(Trusted.RT3; Ubiquitous.RT2)*

• **RD-3.2 Deliver prospective trace capture solutions** that are capable of monitoring development environments, including artifacts and human activities, to infer trace links. *(Ubiquitous.{RT1,RT2,RT3})*

• **RD-3.3 Adopt self-adapting solutions** which are aware of the current project state and reconfigure accordingly in order to optimize the quality of trace links. *(Trusted.RT9)*

5.2 Trace Maintenance

One of the greatest challenges of traceability in practice is that of maintaining trace links as a system evolves. Trace maintenance is essential regardless of whether trace links have been created manually or with tool support. However, while traceability is touted for its ability to support change, the overhead of maintaining trace links can also impede change. Trace Links become stale when source and/or target artifacts are modified without updating impacted trace links. Most industrial requirements management tools address this problem by simply marking the link as *suspect* and, as a result, it is not uncommon to see an industrial trace matrix populated with a high percentage of suspect links.

5.2.1 Evolving Links

The challenge is therefore to evolve trace links automatically as related artifacts change. As shown in our literature study, this area of research has garnered less than one third of the effort ascribed to trace creation problems. One reason for this is the lack of datasets capturing the evolution of real trace links in a project across multiple artifact types (i.e. requirements, design, code, test cases etc.). Without realistic datasets, it is hard for researchers to tackle the problem in a meaningful way.

Prior work in this area has focused on the use of heuristic approaches and trace retrieval methods. Mäder et al. explored the use of heuristic techniques that recognize specific development activities, such as changes applied to a UML model [61], and then evolve trace links according to a set of heuristics. Similar approaches have been explored for changes in requirements [11], and between architectures defined using xADL and source code [70]. Researchers have also developed trace retrieval approaches which attempt to identify deltas between the artifacts retrieved over consecutive traces [92].

One special case of link evolution occurs in product lines which provide a scalable framework for the structured reuse of a wide range of software artifacts, including requirements, architecture, design, code, and test cases [13]. It is potentially cost-effective to reuse trace links across product lines. Several authors have explored the use of traceability to support product line evolution [1], generate trace links across artifacts in a product line [47, 2], and to connect features to artifacts [69]. Given the rapid adoption of product lines in industry, it is essential to investigate and mature this work.

5.2.2 Research Directions

The following research directions address the challenges of link evolution.

• **RD-4.1 Understand patterns of change** across various artifacts including requirements, design, and code. *(None)*

• **RD-4.2 Develop heuristics and probabilistic approaches** for evolving trace links as artifacts change. *(Configurable.RT2)*

• **RD-4.3 Integrate prospective capture with link evolution** techniques. *(None)*

• **RD-4.4 Develop traceability structures to support the evolution of products across a product line**. *(None)*

5.3 Trace Integrity

Trace integrity is concerned with validating the correctness of trace links that have been created and maintained, and/or communicating the quality of an existing set of links. Trace quality is likely to be less than perfect, regardless of whether the trace links have been created by human analysts or by (semi)automated means. It is therefore important for stakeholders to be able to understand the correctness and completeness of the available traceability to assess whether it can be trusted for performing specific tasks.

There are three primary techniques related to trace validation at present (1) eliciting feedback from human analysts, (2) exploiting the semantics and context of each trace link, and (3) computing metrics which serve as indicators of quality. Each of these areas provides potential support for improving and understanding the integrity of trace links, but each also presents its own research challenges.

5.3.1 Improving Integrity through Human Feedback

For many tasks, analysts need to evaluate the generated trace links. Several studies have shown that human feedback is incorrect approximately 25% of the time, which can negatively impact the quality of the generated trace links [53]. Furthermore, the higher the quality of the starting trace matrix, the worse the decisions the analyst makes (and vice versa) [21, 17]). Studies have also shown that humans use different strategies when examining trace links, such as accepting the first good link they find without looking further in the list. Some strategies require high effort while resulting in low accuracy [53].

A limited number of state-of-the-art tracing tools integrate user feedback. The ADAMS [20], POIROT [57], and RETRO [44] research tools collect relevance feedback (i.e. correct or incorrect) on trace links that have been generated automatically using information retrieval techniques. The most common form of feedback uses a standard information technique known as Rocchio which increases or decreases term weightings used to compute similarity scores according to whether a term occurs in a rejected or confirmed trace link [81, 44]. Users may also directly modify the trace query by adding and/or removing terms and the 'before' and 'after' queries can be used to learn a set of query transformation rules which can be used to improve future queries [81, 24].

Progress has been made toward retrieving feedback from

human analysts without deliberate actions on their part [68]. In the traceability domain, researchers have used eye-trackers to explore how analysts verify trace links from requirements to source code. They have also studied the 'delta' tracing process in which prior feedback is assumed to be correct, and only new or modified links are presented to the user. These studies have shown that, while this practice reduces human effort, it negatively impacts the quality of the trace links [92].

Studying the role of the human is important because in many domains, especially safety- and mission-critical ones, results from automated techniques cannot be used unless inspected by a human. We therefore need to gain a better understanding of the process human analysts go through to vet and approve a generated trace link, and to understand which sequences of actions are more or less likely to lead to correct decisions. Furthermore, with the increasing adoption of social collaboration tools, it is interesting to explore the impact of collaborative decision making on the correctness of trace links.

5.3.2 Toward Automating Link Evaluation

Given the potentially large number of trace links in a project, and the cost and effort of validation, it is appealing to consider techniques for assigning confidence scores to each link or to a specified set of links. We see three main areas of work in this area.

First, the semantics of a trace can be analyzed to understand the rationale of the link. Early work in this area has focused on requirements satisfaction [86] and the use of ontology to support the tracing process [41]. Second, the relation between a traced artifact and other artifacts of the same type can be analyzed in order to draw conclusions about the consistency and conclusiveness of existing and missing traces. This compounding evidence can increase confidence in the correctness of a trace link. For example Ghabi and Egyed [33] explored calling dependencies between methods in order to identify regions in the source code that implement a given requirement and found that related requirements were implemented in connected areas of the source code. Kuang et al. [55] extended this idea by demonstrating this type of relationship across both call- and data-dependencies. Third, we need to develop an understanding of how much trust is required based on the intended use of a trace so that the quality of the link can be evaluated within a meaningful context.

5.3.3 Determining and Communicating Confidence

While striving for perfect traceability, it is pragmatic to recognize that as we cannot guarantee complete and accurate traceability, we should devise techniques for clearly communicating confidence levels to the stakeholders. Another interesting research area therefore involves the creation of metrics to evaluate the overall quality of a collection of trace links. Recent work by Rempel et al. generated a TIM from traceability data in a number of safety-critical projects and used formal logic to compare it to the TIM prescribed by relevant process guidelines. They formally defined a number of potential inconsistency and compliance problems and identified the occurrence of such problems in each of the evaluated projects [80]. However, other metrics could be designed to measure factors such as completeness, quality, and/or coverage of existing trace links.

5.3.4 Research Directions

The following research would serve to improve a stakeholders's ability to assess trace integrity.

• **RD-5.1 Develop human-centric tools to support link vetting**. There are many research avenues for supporting the humans in the loop, such as developing expert systems to detect and learn human motion and associated actions/events, customized to each person, or developing recommender and decision support systems (potentially with avatar support) to help human analysts perform vetting and other tracing tasks. *(Trusted.{RT3,RT4,RT6}; Valued.RT1)*

• **RD-5.2 Develop algorithms and supporting tools for automatically evaluating** the correctness of existing trace links, whether created manually or with tool-support. *(Trusted.RT2)*

• **RD-5.3 Create visual dashboards** to visualize the traceability quality of a project. *(Trusted.{RT5,RT8})*

6. USING TRACES

Trace links are created to empower trace users to perform various software engineering tasks more effectively. As such trace usage particularly impacts the goal for traceability to be purposed, i.e. to support stakeholder needs and to be driven by their requirements. From the technical perspective the user interaction layer must provide tools and infrastructure to leverage available traces, and enable project stakeholders to realize the full benefits of available trace data. In this section, we explore research that focuses on enabling stakeholders to (6) access and query trace data, and on (7) presenting trace results for decision-making purposes, through targeted visualizations.

6.1 Accessing and Querying Trace Data

Requirements management tools often provide support for tracing requirements to other artifacts in the software development life-cycle. For example, IBM's RequisitePro allows developers to relate requirements kept within the tool to other tools within the product suite, such as Rational Software Modeler. An all-lifecycle-management (ALM) tool or platform would be an ideal project backbone for through-life traceability, from an academic perspective, as a single repository would keep all the artifact types and link data. However, an industry study showed that many organizations and users prefer chains of task-specific tools to suit their development preferences [9], not necessarily the ones that are part of an ALM solution. Furthermore, organizations are hesitant to be largely dependent upon a single tool vendor by buying into a complete ALM solution. This means that artifacts tend to get scattered across different tools and their individual repositories. This situation makes end-to-end traceability across a development project very difficult to achieve, especially in large projects which span organizational boundaries and have no centralized decision making model. Projects therefore tend to adopt a wide variety of CASE tools across different organizations.

6.1.1 Support for Heterogenity

The ability to access and query heterogeneous trace data is a cross-cutting and enabling requirement for applying end-to-end traceability in real world development contexts. As such, it forms a basis for all seven goals that support the

Traceability Grand Challenge (see Table 1). It is also a prerequisite for making proper use of traceability from a process perspective (see Figure 1). In the technical infrastructure, it impacts the TIM layer (where tracing is planned and supported), the Trace Query Layer (where traces are executed), and the Trace Engine layer (if the decision is made to distribute this across the organization).

The European Computer Manufacturers Association proposes both centralized and distributed reference models for frameworks of software engineering environments [27]. In the centralized model, all artifacts are stored in a common meta-data repository. This is highly attractive in terms of ease of implementation, performance, and security; however, it requires tight integration and coupling between ALM tools. In the less radical distributed model, artifacts reside within their own tool repositories while only meta-data is replicated in a central repository. Several authors have provided prototypical implementations [31, 54]. Kolovos et al. [52] and Barbero et al. [7] describe approaches that merge models on-demand and so propose to create the replicated trace data only when required. A concrete instantiation of the replication approach is proposed by Smith et al. [83]. The authors developed the OPHELIA platform for distributed development tools, which integrates tools by a set of middleware interfaces. Based on this integration layer, the authors propose a generic layer with common services such as notification, knowledge management, metrics calculation, and trace management.

6.1.2 Creating and Reusing Trace Queries

One of the greatest inhibitors of trace usage is the challenge of formulating complex trace queries. Many useful trace queries are quite complex and are not supported by requirements management tools. As a result, users are often required to formulate complicated queries using SQL or other similar languages. This situation inhibits users from using constructed trace links to support tracing tasks [59]. To address these problems, several researchers have explored alternate query languages and query reuse mechanisms. For example, Maletic and Collard [65] describe a Trace Query Language (TQL) which can be used to write trace queries as XPath expressions for artifacts represented in XML format. Mäder et al. propose a visual traceability query notation (VTML) [59] which allows stakeholders of a development project to retrieve relevant traces and the artifacts they are related to more efficiently than standard query notations like SQL. Stakeholders are also enabled to write queries that support their development tasks.

6.1.3 Research Directions

Research here is needed in a couple of directions.

• **RD-6.1 Integrate existing development tools** and other relevant data that is created as part of a development project. The technology should allow flexibility in the ways that tools and data sources can be adapted, including for example, event-driven tool integration, multi-tool repository access, and support for alternative data stores for tools with few interface capabilities. *(Configurable.RT1, Scalable.{RT8,RT10}, Portable.RT3)*

• **RD-6.2 Provide intuitive forms of query mechanism** that do not require specialized training. These may include simpler techniques for formulating new queries or retrieving existing, reusable, trace queries. *(Scalable.RT2)*

6.2 Visualizing Trace Data

Traceability is put in place to establish useful links between the artifacts of a development process, so that the eventual traversal of these links can support various engineering tasks. At present, trace information can be difficult for developers to use, since little research attention has been directed toward its presentation and eventual end-use. The most common way to enter and show traceability information in practice remains the trace matrix [56], a two-dimensional view on to a multidimensional information space, despite the well-known difficulties associated with its scalability. Reasons for the persistence of trace matrices are clear: they are simple to create and use when kept to a manageable size and when used to associate two types of artifact; they can be constructed manually or using general purpose tools such as spreadsheets; they are intuitive for non experts to create and understand; and they remain the dominant representation provided for traceability in most commercial tools. More generally, linear, tabular, hierarchical, and graph-based representations are the most prevalent forms of visualization used to depict traceability information in commercial tool support, and hyper-links (cross-references) are routinely used to associate artifacts and traverse the links interactively.

6.2.1 Trace Visualization

Enormous advances have been made in popular techniques and tools for information and knowledge visualization [42]. Visual analytics are now a common form of support for decision-making activities in many fields of endeavor [87] and advice on selecting suitable visualization techniques is readily available [50, 82, 76]. Despite some pockets of research, our field has been slow to keep pace, and now needs to re-examine its information-seeking needs and mechanisms more systematically. As projects continue to grow in size, duration and complexity, researchers have strived to tackle the dimensions of scale and address its resulting visual clutter through augmenting and combining well-known visualization techniques, and developing new ones. Tackling scale head-on, Chen et al. [10] propose combining treemap and hierarchical tree visualization techniques to show both the global structure of traces and a detailed overview of each trace. Merten et al. [67] propose interactive Sunburst and Netmap visualizations as a way to achieve similar goals. Addressing other dimensions of scale, three-dimensional visualizations have been developed to depict temporal traceability information [4], and to convey traceability across modelling abstraction levels [90]. However, for the fruits of this, and other related research, to move from prototypes to commercial tools, evidence of effectiveness in practice is now needed.

6.2.2 Selecting Trace Visualizations

Concurrent with the development and evolution of techniques, traceability researchers have emphasized the need to better understand the users, tasks, and project constraints that drive the development and selection of suitable visualizations for traceability purposes [66, 39, 91, 16]. Future tools will no doubt need to provide a collection of visualizations to support different users and tasks, and these will need to span all areas of an end-to-end traceability process. To this end, visualization is now finding its way into traceability strategizing tasks, where a visual language is proposed for defining the traceability strategy on a project [59].

A recent empirical study examined four common visualizations used to present traceability information (matrices, graphs, lists, and hyperlinks), to investigate which ones were better suited to which tasks [56]. It found that matrices and graphs were preferred to support management tasks, while hyperlinks were preferred to support implementation and testing tasks. Matrices were found appropriate to gain an overview of the traceability, while graphs were found suited to navigating the resulting traces.

Such studies bring an important focus to task-driven visualization, the longer-term objective of this research being to suggest the most appropriate traceability visualization(s) for any task at hand. To achieve this goal, Niu et al. [72] developed a framework for assessing the effectiveness of visualizations for supporting decision-making activities, based upon how users interact with the visualization to augment their knowledge, and then used the framework to critique existing visualizations and propose the principled development of new ones.

6.2.3 Research Directions

We envisage a future in which there are better visual ways to interactively define, create, maintain, analyze, and use traceability information effectively. Necessary research comprises three directions.

• **RD-7.1 Construct a taxonomy of available visualizations and fundamental traceability tasks**. Bridge these by exploring the basic visualization principles that they either provide or demand. *(Scalable.RT2)*

• **RD-7.2 Gather and share user-based empirical data** to evaluate trace visualizations and direct the formulation of novel ones. *(Purposed.RT6)*

• **RD-7.3 Perform in-situ user studies to evaluate and understand task-specific needs for trace information** and develop novel ways to provide the needed information to stakeholders. *(Purposed.RT1)*

7. CONCLUSIONS

In this paper, we have identified seven research areas and their associated "research directions" which must be addressed in order to achieve the grand challenge of ubiquitous traceability. Each of these research directions is fully actionable and will help our community to work collaboratively towards advancing the state-of-the-art and state of practice in traceability.

The identified research directions are quite varied in nature. Some focus on algorithmic solutions, others on process improvement, and still others on technical infrastructure needs. This suggests that advancing the state-of-the-art in software traceability will require the cooperation of researchers with different skill-sets from areas as diverse as information systems, data mining, visualization, and systems engineering. Furthermore, traceability research addresses a current and pressing need in industry, and therefore also requires the engagement of industry practitioners. Case studies exploring successes and failures in software traceability practice keep researchers focused on the real problems to be addressed, while academic-industry collaborations provide invaluable opportunities for exploring the application of mature research in an industrial context.

Roadmap-style papers have traditionally provided directions for future research while social computing infrastructures provide a more interactive framework for supporting research collaborations and ongoing discussions. To this end, we have established an online *Research Directions* forum at CoEST.org.

Finally, wherever appropriate, we also provide downloadable, executable, baselined experiments, developed in *TraceLab* [49]. TraceLab is an innovative experimental environment which allows researchers to reuse, reproduce, and/or modify previous experiments, compose new experiments from a combination of existing and user-defined components, use publicly available datasets, exchange components, and comparatively evaluate results against previous benchmarks. We provide links to these experiments from the Research Directions forum to encourage comparative envaluation and reproducibility of experimental results.

We encourage researchers to actively engage in the ongoing discussion, to keep the community informed of important advances and new challenges as they emerge, and where possible to use existing experimental baselines or create new ones for others to use. Our hope is that we can engage in a lively, ongoing, and interactive discussion centered around the identified research directions and work together towards achieving our goal of ubiquitous traceability.

Acknowledgments

Work on this paper was partially funded by NSF Grants CCF-1319680 (Huang), CNS-0959924 (Hayes,Huang), and the German BMBF grant 16V0116 (Mäder).

8. REFERENCES

[1] S. Ajila and A. B. Kaba. Using traceability mechanisms to support software product line evolution. In *IEEE International Conference on Information Reuse and Integration (IRI)*, pages 157–162, 2004.

[2] N. Anquetil, U. Kulesza, R. Mitschke, A. Moreira, J.-C. Royer, A. Rummler, and A. Sousa. A model-driven traceability framework for software product lines. *Software and System Modeling*, 9(4):427–451, 2010.

[3] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo. Recovering traceability links between code and documentation. *IEEE Transactions on Software Engineering*, 28(10):970–983, 2002.

[4] G. Antoniol, E. Merlo, Y.-G. Guéhéneuc, and H. Sahraoui. On feature traceability in object oriented programs. In *3rd International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE)*, pages 73–78, 2005.

[5] P. Arkley and S. Riddle. Overcoming the traceability benefit problem. In *13th IEEE International Requirements Engineering Conference (RE)*, pages 385–389, 2005.

[6] H. U. Asuncion, A. Asuncion, and R. N. Taylor. Software traceability with topic modeling. In *32nd ACM/IEEE International Conference on Software Engineering (ICSE)*, pages 95–104, 2010.

[7] M. Barbero, M. Didonet, D. Fabro, and J. Bézivin. Traceability and provenance issues in global model management. In *3rd ECMDA-Traceability Workshop (ECMDA-TW)*, 2007.

[8] F. Blaauboer, K. Sikkel, and M. N. Aydin. Deciding to adopt requirements traceability in practice. In *19th International Conference on Advanced Information Systems Engineering (CAiSE)*, pages 294–308, 2007.

[9] E. Bouillon, P. Mäder, and I. Philippow. A survey on usage scenarios for requirements traceability in practice. In *19th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ)*, volume 7830 of *LNCS*, pages 158–173, 2013.

[10] X. Chen, J. G. Hosking, and J. Grundy. Visualizing traceability links between source code and documentation. In *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 119–126, 2012.

[11] J. Cleland-Huang, C. K. Chang, and Y. Ge. Supporting event based traceability through high-level recognition of change events. In *26th International Computer Software and Applications Conference (COMPSAC)*, pages 595–602, 2002.

[12] J. Cleland-Huang, M. Heimdahl, J. Huffman Hayes, R. Lutz, and P. Mäder. Trace queries for safety requirements in high assurance systems. In *18th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ)*, volume 7195 of *LNCS*, pages 179–193, 2012.

[13] P. C. Clements and L. Northrop. *Software Product Lines: Practices and Patterns*. SEI Series in Software Engineering. Addison-Wesley, 2001.

[14] CoEST: Center of excellence for software traceability, http://www.CoEST.org.

[15] Committee for Advancing Software Intensive Systems Producibility. *Critical Code: Software Producibility for Defense*. The National Academies Press, 2011.

[16] J. R. Cooper, S.-W. Lee, R. A. Gandhi, and O. Gotel. Requirements engineering visualization: A survey on the state-of-the-art. In *4th International Workshop on Requirements Engineering Visualization (REV)*, pages 46–55, 2009.

[17] D. Cuddeback, A. Dekhtyar, J. Huffman Hayes, J. Holden, and W.-K. Kong. Towards overcoming human analyst fallibility in the requirements tracing process: Nier track. In *33rd International Conference on Software Engineering (ICSE)*, pages 860–863, 2011.

[18] D. C. de Leon and J. Alves-Foss. Experiments on processing and linking semantically augmented requirement specifications. In *37th Hawaii International Conference on System Sciences (HICSS)*, 2004.

[19] A. De Lucia, F. Fasano, R. Oliveto, and G. Tortora. Enhancing an artefact management system with traceability recovery features. In *20th IEEE International Conference on Software Maintenance (ICSM)*, pages 306–315, 2004.

[20] A. De Lucia, R. Oliveto, and G. Tortora. Adams re-trace. In *35th International Conference on Software Engineering (ICSE)*, pages 839–842, 2008.

[21] A. Dekhtyar, O. Dekhtyar, J. Holden, J. Huffman Hayes, D. Cuddeback, and W.-K. Kong. On human analyst performance in assisted requirements tracing: Statistical analysis. In *19th IEEE International Requirements Engineering Conference (RE)*, pages 111–120, 2011.

[22] A. Dekhtyar, J. Huffman Hayes, S. K. Sundaram, E. A. Holbrook, and O. Dekhtyar. Technique integration for requirements assessment. In *15th IEEE International Requirements Engineering Conference (RE)*, pages 141–150, 2007.

[23] A. Delater and B. Paech. Analyzing the tracing of requirements and source code during software development - a research preview. In *19th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ)*, volume 7830 of *LNCS*, pages 308–314, 2013.

[24] T. Dietrich, J. Cleland-Huang, and Y. Shin. Learning effective query transformations for enhanced requirements trace retrieval. In *28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 586–591, 2013.

[25] B. Dit, M. Revelle, and D. Poshyvanyk. Integrating information retrieval, execution and link analysis analysis algorithms to improve feature location in software. *Empirical Software Engineering*, 18(2):277–309, 2013.

[26] A. Egyed, P. Grünbacher, M. Heindl, and S. Biffl. Value-based requirements traceability: Lessons learned. In *15th IEEE International Requirements Engineering Conference (RE)*, pages 115–118, 2007.

[27] European Computer Manufacturers Association. Reference model for frameworks of software engineering environments. Technical Report TR/55, NIST Special Publication 500–211, ECMA, Geneva, Switzerland, June 1993.

[28] D. Falessi, G. Cantone, and G. Canfora. Empirical principles and an industrial case study in retrieving equivalent requirements via natural language processing techniques. *IEEE Transactions on Software Engineering*, 39(1):18–44, 2013.

[29] Federal Aviation Authority (FAA). *DO-178B: Software Considerations in Airborne Systems and Equipment Certification*, faa's advisory circular ac20-115b edition.

[30] Food and Drug Administration. *Guidance for the Content of Premarket Submissions for Software Contained in Medical Devices*, 2005.

[31] R. Freude and A. Königs. Tool integration with consistency relations and their visualization. In *ESEC/FSE Workshop on Tool Integration in System Development*, pages 6–10, 2003.

[32] M. Gethers, R. Oliveto, D. Poshyvanyk, and A. De Lucia. On integrating orthogonal information retrieval methods to improve traceability link recovery. In *27th IEEE International Conference on Software Maintenance (ICSM)*, pages 133–142, 2011.

[33] A. Ghabi and A. Egyed. Code patterns for automatically validating requirements-to-code traces. In *27th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 200–209, 2012.

[34] O. Gotel, J. Cleland-Huang, J. Huffman Hayes, A. Zisman, A. Egyed, P. Grünbacher, and G. Antoniol. The quest for ubiquity: A roadmap for software and systems traceability research. In *21st IEEE International Requirements Engineering*

Conference (RE), pages 71–80, 2012.

[35] O. Gotel, J. Cleland-Huang, J. Huffman Hayes, A. Zisman, A. Egyed, P. Grünbacher, A. Dekhtyar, G. Antoniol, and J. Maletic. The grand challenge of traceability (v1.0). In J. Cleland-Huang, O. Gotel, and A. Zisman, editors, *Software and Systems Traceability*, pages 343–409. Springer, 2012.

[36] O. Gotel, J. Cleland-Huang, J. Huffman Hayes, A. Zisman, A. Egyed, P. Grünbacher, A. Dekhtyar, G. Antoniol, J. Maletic, and P. Mäder. Traceability fundamentals. In J. Cleland-Huang, O. Gotel, and A. Zisman, editors, *Software and Systems Traceability*, pages 3–22. Springer, 2012. 10.1007/978-1-4471-2239-51.

[37] O. Gotel and A. Finkelstein. An analysis of the requirements traceability problem. In *1st IEEE International Conference on Requirements Engineering (ICRE)*, pages 94–101, 1994.

[38] O. Gotel and A. Finkelstein. Contribution structures [requirements artifacts]. In *2nd IEEE International Symposium on Requirements Engineering (RE)*, pages 100–107, 1995.

[39] O. Gotel, F. T. Marchese, and S. J. Morris. On requirements visualization. In *2nd International Workshop on Requirements Engineering Visualization (REV)*, page 11, 2007.

[40] S. Guckenheimer and J. Perez. *Software Engineering with Microsoft Visual Studio Team System*. Addison Wesley, 2006.

[41] J. Guo, J. Cleland-Huang, and B. Berenbach. Foundations for an expert system in domain-specific traceability. In *21st IEEE International Requirements Engineering Conference (RE)*, pages 42–51, 2013.

[42] J. Heer, M. Bostock, and V. Ogievetsky. A tour through the visualization zoo. *Commun. ACM*, 53(6):59–67, June 2010.

[43] E. A. Holbrook, J. Huffman Hayes, A. Dekhtyar, and W. Li. A study of methods for textual satisfaction assessment. *Empirical Software Engineering*, 18(1):139–176, 2013.

[44] J. Huffman Hayes, A. Dekhtyar, and S. K. Sundaram. Advancing candidate link generation for requirements tracing: The study of methods. *IEEE Transactions on Software Engineering*, 32(1):4–19, 2006.

[45] E. Hull, K. Jackson, and J. Dick. *Requirements Engineering*. Springer, 2002.

[46] C. Ingram and S. Riddle. Cost-benefits of traceability. In *Software and Systems Traceability*, pages 23–42. Springer, 2012.

[47] W. Jirapanthong and A. Zisman. Xtraque: traceability for product line systems. *Software and System Modeling*, 8(1):117–144, 2009.

[48] V. Katta and T. Stälhane. Traceability of safety systems: approach, meta-model and tool support,. Tech. report hwr-1053, oecd halden reactor project, Institute for Energy Technology Note: Available upon request., 2012.

[49] E. Keenan, A. Czauderna, G. Leach, J. Cleland-Huang, Y. Shin, E. Moritz, M. Gethers, D. Poshyvanyk, J. Maletic, J. Huffman Hayes, A. Dekhtyar, D. Manukian, S. Hossein, and D. Hearn. Tracelab: An experimental workbench for equipping researchers to innovate, synthesize, and comparatively evaluate traceability solutions. In *Tool Demo, 34th International Conference on Software Engineering (ICSE)*, pages 1375–1378, 2012.

[50] P. R. Keller and M. M. Keller. *Visual Cues: Practical Data Visualization*. IEEE Computer Society Press, 1994.

[51] V. Kirova, N. Kirby, D. Kothari, and G. Childress. Effective requirements traceability: Models, tools, and practices. *Bell Labs Technical Journal*, 12(4):143–157, 2008.

[52] D. S. Kolovos, R. F. Paige, and F. A. C. Polack. C.: On-demand merging of traceability links with models. 3 rd ecmda traceability workshop, 2006.

[53] W.-K. Kong, J. Huffman Hayes, A. Dekhtyar, and O. Dekhtyar. Process improvement for traceability: A study of human fallibility. In *20th IEEE International Requirements Engineering Conference (RE)*, pages 31–40, 2012.

[54] A. Königs and A. Schürr. Mdi: A rule-based multi-document and tool integration approach. *Software and System Modeling*, 5(4):349–368, 2006.

[55] H. Kuang, P. Mäder, H. Hu, A. Ghabi, L. Huang, J. Lv, and A. Egyed. Do data dependencies in source code complement call dependencies for understanding requirements traceability? In *28th IEEE International Conference on Software Maintenance (ICSM)*, pages 181–190, 2012.

[56] Y. Li and W. Maalej. Which traceability visualization is suitable in this context? a comparative study. In *18th International Conference on Requirements Engineering: Foundation for Software Quality (REFSQ)*, pages 194–210, 2012.

[57] J. Lin, C. C. Lin, J. Cleland-Huang, R. Settimi, J. Amaya, G. Bedford, B. Berenbach, O. Ben Khadra, C. Duan, and X. Zou. Poirot: A distributed tool supporting enterprise-wide automated traceability. In *14th IEEE International Requirements Engineering Conference (RE)*, pages 356–357, 2006.

[58] S. Lohar, S. Amornborvornwong, A. Zisman, and J. Cleland-Huang. Improving trace accuracy through data-driven configuration and composition of tracing features. In *9th Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*, pages 378–388, 2013.

[59] P. Mäder and J. Cleland-Huang. A visual language for modeling and executing traceability queries. *Software and System Modeling*, 12(3):537–553, 2013.

[60] P. Mäder and A. Egyed. Assessing the effect of requirements traceability for software maintenance. In *28th IEEE International Conference on Software Maintenance (ICSM)*, pages 171–180, 2012.

[61] P. Mäder and O. Gotel. Towards automated traceability maintenance. *Journal of Systems and Software*, 85(10):2205–2227, 2012.

[62] P. Mäder, O. Gotel, and I. Philippow. Getting back to basics: Promoting the use of a traceability information model in practice. In *5th Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE)*, 2009.

[63] P. Mäder, O. Gotel, and I. Philippow. Motivation matters in the traceability trenches. In *17th IEEE*

International Requirements Engineering Conference (RE), pages 143–148, 2009.

[64] P. Mäder, P. L. Jones, Y. Zhang, and J. Cleland-Huang. Strategic traceability for safety-critical projects. *IEEE Software*, 30(3):58–66, 2013.

[65] J. I. Maletic and M. L. Collard. Tql: A query language to support traceability. In *5th Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE)*, pages 16–20, 2009.

[66] A. Marcus, X. Xie, and D. Poshyvanyk. When and how to visualize traceability links? In *3rd International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE)*, pages 56–61, 2005.

[67] T. Merten, D. Juppner, and A. Delater. Improved representation of traceability links in requirements engineering knowledge using sunburst and netmap visualizations. In *4th Workshop Managing Requirements Knowledge (MARK)*, pages 17–21, 2011.

[68] T. Miller and S. Agne. Attention-based information retrieval using eye tracker data. In *3rd International Conference on Knowledge Capture (K-CAP)*, pages 209–210, 2005.

[69] R. Mitschke and M. Eichberg. Supporting the evolution of software product lines. In *4th ECMDA-Traceability Workshop (ECMDA-TW)*, pages 87–96, 2008.

[70] L. G. P. Murta, A. van der Hoek, and C. M. L. Werner. Archtrace: Policy-based support for managing evolving architecture-to-implementation traceability links. In *21st IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 135–144, 2006.

[71] P. Nistala and P. Kumari. Establishing content traceability for software applications: An approach based on structuring and tracking of configuration elements. In *7th Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE)*, 2013.

[72] N. Niu, S. Reddivari, and Z. Chen. Keeping requirements on track via visual analytics. In *21st IEEE International Requirements Engineering Conference (RE)*, pages 205–214, 2013.

[73] S. C. of RTCA. DO-178C, software considerations in airborne systems and equipment certification, 2011.

[74] A. Panichella, B. Dit, R. Oliveto, M. Di Penta, D. Poshyvanyk, and A. De Lucia. How to effectively use topic models for software engineering tasks? an approach based on genetic algorithms. In *35th International Conference on Software Engineering (ICSE)*, pages 522–531, 2013.

[75] M. Panis. Successful deployment of requirements traceability in a commercial engineering organization...really. In *18th IEEE International Requirements Engineering Conference (RE)*, pages 303–307, 2010.

[76] D. Pfitzner, V. Hobbs, and D. Powers. A unified taxonomic framework for information visualization. In *Asia-Pacific Symposium on Information Visualisation (APVis)*, pages 57–66, 2003.

[77] D. Poshyvanyk. Using information retrieval to support software maintenance tasks. In *25th IEEE International Conference on Software Maintenance (ICSM)*, pages 453–456, 2009.

[78] B. Ramesh and M. Jarke. Toward reference models of requirements traceability. *IEEE Transactions on Software Engineering*, 27(1):58–93, 2001.

[79] P. Rempel, P. Mäder, and T. Kuschke. An empirical study on project-specific traceability strategies. In *21st IEEE International Requirements Engineering Conference (RE)*, pages 195–204, 2013.

[80] P. Rempel, P. Mäder, T. Kuschke, and J. Cleland-Huang. Mind the gap: Assessing the conformance of software traceability to relevant guidelines. In *36th International Conference on Software Engineering (ICSE)*, 2014.

[81] Y. Shin and J. Cleland-Huang. A comparative evaluation of two user feedback techniques for requirements trace retrieval. In *27th Annual ACM Symposium on Applied Computing (SAC)*, pages 1069–1074, 2012.

[82] B. Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *IEEE Symposium on Visual Languages (VL)*, pages 336–343, 1996.

[83] M. Smith, D. Weiss, P. Wilcox, and R. Dewar. The ophelia traceability layer. In F. Angeli, editor, *Cooperative Methods and tools for distributed software processes*, volume 380.222 of *RCOST/SOFTWARE TECHNOLOGY*, pages 88–464, 2003.

[84] G. Spanoudakis, A. Zisman, E. Pérez-Miñana, and P. Krause. Rule-based generation of requirements traceability relations. *Journal of Systems and Software*, 72(2):105–127, 2004.

[85] H. Sultanov, J. Huffman Hayes, and W.-K. Kong. Application of swarm techniques to requirements tracing. *Requirements Engineering*, 16(3):209–226, 2011.

[86] S. K. Sundaram, J. H. Hayes, A. Dekhtyar, and E. A. Holbrook. Assessing traceability of software engineering artifacts. *Requirements Engineering*, 15(3):313–335, 2010.

[87] J. J. Thomas and K. A. Cook. A visual analytics agenda. *IEEE Computer Graphics and Applications*, 26(1):10–13, 2006.

[88] E. Vassev and M. Hinchey. Autonomy requirements engineering. *IEEE Computer*, 46(8):82–84, 2013.

[89] A. von Knethen. Change-oriented requirements traceability: Support for evolution of embedded systems. In *18th IEEE International Conference on Software Maintenance (ICSM)*, pages 482–485, 2002.

[90] J. von Pilgrim, B. Vanhooff, I. Schulz-Gerlach, and Y. Berbers. Constructing and visualizing transformation chains. In *4th European Conference on Model Driven Architecture – Foundations and Applications (ECMDA-FA)*, pages 17–32, 2008.

[91] S. Winkler. On usability in requirements trace visualizations. In *3rd International Workshop on Requirements Engineering Visualization (REV)*, pages 56–60, 2008.

[92] S. Winkler. Trace retrieval for evolving artifacts. In *5th Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE)*, pages 49–56, 2009.