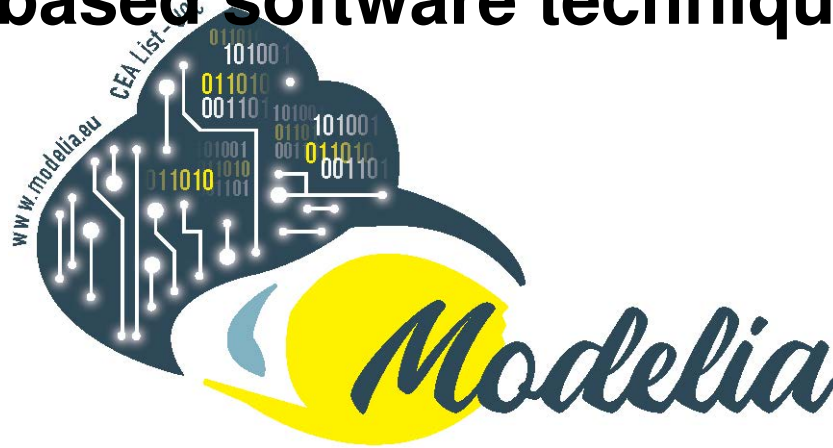




R&I

Universitat Oberta
de Catalunya

1st Deliverable: Survey of traceability techniques with a focus on their applications in AI-based software techniques



Edouard Batot

SOM Research Lab. Office 223
Av. Carl Friedrich Gauss, 5. Building B3
08860 Castelldefels (Barcelona), Spain

SOM

CONTENTS

1	Introduction	5
2	Technical report	7
2.1	Context	7
2.2	State of the art	8
2.3	Toward a common standard	10
2.3.1	Different interpretations of the traceability concept	10
2.3.2	Key traceability terms' definitions	11
2.3.3	Traceability metamodel	13
2.4	Research questions	14
2.4.1	RQ1: How are traces represented?	14
2.4.2	RQ2: How are traces identified?	14
2.4.3	RQ3: What tool support is available to manage and apply traces?	14
2.5	Survey method	15
2.5.1	Data source and search strategy	15
2.5.2	Pruning	16
2.5.3	Snowballing	16
2.5.4	Threads to validity in the selection process	17
2.6	Traceability, a feature model	18
2.6.1	Trace definition and representation	18
2.6.2	Trace identification	22
2.6.3	Trace management	24
2.7	AI Explainability as a purpose to traceability	26
2.7.1	Survey method	27
2.7.2	Survey analysis	28
2.8	Conclusion	28

LIST OF ACRONYMS

MDE	Model-driven Engineering
UML	Unified Modeling Language
TIM	Traceability Information Model
RE	Requirement Engineering
AI	Artificial Intelligence
ML	Machine Learning
EA	Evolutionary Algorithm
ANN	Artificial Neural Network
NLP	Natural Language Processing
VSM	Vector Space Model
LSI	Latent Semantic Indexing
LSTM	Long Short-Term Memory
SOM	Systems, Software and Models Lab
UOC	Open University of Catalonia
CEA	Commissariat à l'énergie atomique et aux énergies alternatives

1 INTRODUCTION

The purpose of this report is to document the work done as part of the First Task described in the Convention de l'équipe de recherche commune. This document corresponds to Deliverable 1. The content of this document is a scientific report on the conjunct use of traceability, modelling, and AI, with a scrutiny on the potential such techniques could bring to AI-enabled system architecture.

The content of this document is a scientific submission in the form of the description of the state-of-the-art, a linguistic metamodel, and a feature model of traceability research and industrial applications.

2 TECHNICAL REPORT

2.1 CONTEXT

In the last decade we have acknowledged a tremendous rise of artificial intelligence (AI) techniques in our daily life experience and software systems are not an exception.

If the introduction of AI in software components (and even in the development of such components) has become widespread, it has also introduced a new degree of uncertainty as to the potential variations in their behaviour [38]. Indeed, better solutions must be developed to achieve a high-quality AI-enabled software [57], mainly related to reduce the opacity of AI Components in favour of a better transparency and explainability of their decisions.

This need has always been important in software and systems development as well. Across the years, there has always been an interest in developing techniques to facilitate the representation and analysis of traces and links between related artefacts to help explaining their execution and evolution.

The importance of traceability has been largely recognized, especially in systems engineering and it became a primary concern to commit to certification authorities in all commercial software-based aerospace systems with the RTCA DO-178C (2012) [51]. The consideration of various levels of abstraction in software development and the meaning of verification in model-based development paradigm - which figures abstract representations (models) as the core artefact for conceptualization - was latter introduced with companion documents (Specifically, DO-331). The automotive industry has followed the same path with the construction of an international standard for functional safety [40]. At code, requirement, model, human and organisational level, a transversal view provided through traces of processes and artefacts executions figures an ideal for software development and maintenance.

Traceability is less considered in software engineering approaches. Despite these important evidences on the need for explicit (and automated) tracing abilities in software development, traceability is not widely adopted, even less automated and there is little feedback from its concrete use in industry [63]. Winkler *et al.* [87] show all the disregard such "mandatory" characteristic implies - expressly to safety critical software demanding a legal assessment or certification to run [87], and Mader *et al.* show that the benefits of automated traceability is still to be justified to stakeholders who still prefer the *ad hoc* manual burden [46]. As a matter of fact, authors publishing a guide to the software engineering body of knowledge only mention traceability without in-depth reflection [14]. With no standard definition or representation of traces , a gap remains between broader fields of investigation beneficial from automated tracing ability (*e.g.*, requirement engineering, program understanding, testing). Many studies suggest that there is a lack of synthetic investigations on traceability and re-

2.2 State of the art

search on traceability is locked into sub-fields. For this lack of communication tools sharing between research teams remains insufficient [5, 87, 88]. [68]

Notwithstanding these limitations, we see a lot of potential in the contribution of traceability techniques to software development. Even more in a context where AI techniques are being integrated in such development processes and therefore explainability concerns are increasing. This paper aims to first provide a comprehensive survey of the state of the art of traceability techniques in software development and their limitations across several dimensions (trace representation, identification, management,...). To help compare current and future traceability proposals we also provide a feature model that aims to organize and structure the main traceability concepts. Next, we conducted a second survey more focused on the use of traceability in the new breed of AI techniques being integrated in software development. Here, the goal was to see how large was the opportunity of (and the need for) traceability in this growing field.

The paper is organised as follows. After a brief introduction, we present in Section 2.2 a synthesis of the state of the art in traceability research. We then introduce basic concept required to the remaining of the paper in Section 2.3. We establish research questions and describe our survey method in Sections 2.4 and 2.5. We answer the research questions with a detailed feature model grounded on the analysis of the survey results in Sections 2.6. We question whether traceability is used for AI explainability in software engineering applications in Section 2.7 before we conclude.

2.2 STATE OF THE ART

We present and discuss in this section works related to ours to motivate and justify the need of a new survey in this field.

An initial survey of the work done to increase the automation of traceability in software systems was published in 2007 with a great impact [17]. Authors describe nine best practices to augment the potential of traceability that will remain until today the long reach goal to achieve in the emerging research community. They call for a clear establishment of traceability purposes. They also call for an effort from researchers to define the granularity they need to achieve their goals. Researchers must as well keep in mind what will require the integration of this ideal tooling into native software environments. On a concrete level, requirement traceability needs quality specifications and exact term definitions to express clearly the conceptualization occurring among them. This requires rich content. Rich enough to bridge the cognitive gaps that plague transitions from one lifecycle phase to the next. Rich enough as well to connect intra-domain semantic synonyms and keep a coherent hierarchy between traced concepts. The same year, Galvao *et al.* surveys approaches addressing MDE limitations regarding traceability [26]. In this work, authors show the enormous potential of MDE to

represent traces, to generate them automatically, and to assist their management and maintenance. MDE is a suitable paradigm to reach for the end-to-end traceability ideal. Authors describe in great details on the usages of traceability in MDE in a distinctive comparison of the different existing approaches. They conclude that a lack in design and specification modeling, as well as a lack in traceability metamodeling, hampers the automation of traceability techniques through the software life cycle in the MDE paradigm.

These synthetic works preceded a peak in the number of investigations mentioning explicitly traceability. (In Fig. 3, we see that in 2010 22 studies mentioned traceability and modeling.) Among them, Winkler *et al.* published an in-depth exploration of both traceability in requirement engineering and model-based development [87]. Authors point out that researchers in the two research fields do not communicate enough between each others. This hampers the establishment of common rich and ubiquitous traceability techniques. The authors points out at a lack of tooling in a field where everything is set to support automated traceability. Automated transformations are the very best soil to grow end-to-end traceability at a core level. Yet, even in the modeling community, traceability practices are consider far from mature. There is ongoing research to model the traceability process, but there lacks a pro-active sharing between the larger communities researchers belong to (such as requirement engineering, program understanding, and so on). This piece of work is the closest to ours and we aim at continuing this work with taking account of new trends and describing a more functional approach.

Since studies dedicated to traceability come from specific areas, the general understanding is long to converge. All and every literature synthesis call for more common terminology to allow the sharing of scientific results and help augment the maturity of the emerging research field [87]. On an industrial perspective, some industrial interviewees did not even know what traceability actually is about and rose concerns about its salience during the development process [18, 83]. Traceability is a major concern in the development of safety critical system, but since it is mandatory it does not gain much enthusiasm. As Mader *et. al.* show in their work, traceability has been considered poorly (by non-traceability stakeholders and researchers) and would benefit a better understanding of its purposes. This would help gather evidences (if they still lack) of the importance of tracing abilities during software development and maintenance [46].

Based on these previous works and the disruptive popularity of AI in software engineering, we believe there is a need for a survey that: 1- summarizes and reflects on the changes and new proposals in the last decade and 2 - goes beyond describing the proposals and makes an effort to synthesize their information in a common traceability glossary, meta-model and feature model to help better characterize the area, application scenarios and potential impact on the growing field of AI-enabled software engineering.

2.3 TOWARD A COMMON STANDARD

The ubiquitous presence of traceability in software literature spreading through out different fields and application scenarios has hampered the use of a consistent and shared terminology around the key traceability concepts¹. In this perspective, we argue that the "incoherency problem" arises in traceability research [86]. Even if an individual article made a claim that withstood rigorous testing and statistical analysis, it might not use the same words as an adjacent article, or it would use the same words but intend different meanings [5, 13]. For instance, the term *traceability* is used to designate both the ability to trace system elements, and the traceability links (the relations) themselves. Regarded as a mere mandatory constraint in safety critical domains or as a lucky by-product of model transformation execution in automated model manipulation, no *de-facto* or *de-juro* standard rules the landscape of traceability investigations [46].

In this section, we first review the different usages of traceability in the literature and then we integrate them in a single metamodel and set of term definitions that we believe will help to understand and compare the different traceability proposals we will be analyzing later on.

2.3.1 DIFFERENT INTERPRETATIONS OF THE TRACEABILITY CONCEPT

Traceability research was born with a definition coined by Gotel *et al.*: "Requirements traceability refers to the ability to describe and follow the life of a requirement, in both a forwards and backwards direction", a broad definition focused on the ability to link requirements to code or design elements of a system [29]. Decades later, traceability still refers mainly to *requirement-driven approaches* that aim at understanding the semantics of requirements [8, 13]. While approaches vary in their means and goals, the attention remains primarily focused on the dependencies linking specifications to design and code artefacts.

Similarly, *modeling approaches* see traceability techniques as useful tools aimed at linking artefacts at different levels of abstraction (and different phases and increment of the software life cycle) [47, 81], and *transformation approaches* make profit of the use of automated transformations to generate (and maintain) automatically trace artefacts [26]. Others use *feature traceability* to refer to the ability to locate features in software artefacts [50].

In all cases, we notice a division of knowledge into four main areas: i) strategizing traceability, ii) the creation, maintenance and evolution of trace artefacts, iii) the identification and record of trace links, and iv) the usability of trace representations [5, 87]). The first is higher level concern, the three others are first order features of tracing ability.

¹A trace-based paper was awarded the most influential paper in the past 10 years at one of the main venue on software engineering (ICSE) [43]. The work introduced a novel trace-based approach to debugging that prominent industrials made use of with brio. Yet, if debugging has made a buzz, traceability remained a secondary concern. (Even though "trace" is mention 46 times in the 10 pages paper).

- **Strategizing traceability** means defining an explicit purpose to a traceability approach including the means to achieve it and a way to measure the (remaining) distance to that goal.
- **Trace artefact management** means the definition of trace artefact representations, their characteristics, design, and variability. The emphasis is here put on the language definition with metamodels and frameworks, the typing of the links, and the maintenance of the coherence in traceability artefacts. In this area, model artefacts and traces co-evolution is an important concern in the model-driven paradigm.
- **Trace link identification** designates the identification of traces in a software system, be it a post-requirement assisted elicitation, or a live record during a system execution. Most of the literature in this area focus on the automation of the identification process by means of machine learning techniques.
- **Trace usability** means ways to make use of existing traces. This includes the tool support for automated task on existing trace artefacts such as the persistence, retrieval and analysis.

2.3.2 KEY TRACEABILITY TERMS' DEFINITIONS

We propose some general definitions for the most frequently encountered terms while searching for and studying solutions for traceability in any of the above categories. These definitions aim to encompass and unify all the different uses and vision of traceability depicted above.

Note that these definitions (same for the metamodel following up in the next subsection) do not aim to cover all the different visions on traceability but provide a common core generic enough to be then adapted to specific scenarios. This is why we do this work before conducting the survey as this is a preliminary step, based on our knowledge of the field, to help us classify the survey results and not a result of the survey itself.

- **Traceability** is the ability to trace different artefacts of a system (of systems). It is defined in the IEEE Standard Glossary of Software Engineering Terminology [54] as
 1. The degree to which a relationship can be established between two or more products of the development process, especially products having a predecessor–successor or master–subordinate relationship to one another. [...]
 2. The degree to which each element in a software development product establishes its reason for existing.
- A **trace** is a path from one artefact to another. Composed of atomic **links** that directly relate artefacts with each others. The representation of traces, their data structure

2.3 Toward a common standard

and behaviour, is defined in a **traceability information model** (TIM), or traceability metamodel [21]. In both cases it defines at the language level the concepts and relationships available for tracing. With the manifolds of traceability purposes, no common **representation** has emerged yet.

- A **trustee** is the **human** actor responsible for an artefact.
- A **link** is a relationship between two artefacts. The **type of the relationship** informs about its semantics. Typing is a primary concern in conceptual modeling in general [55]. The heterogeneous nature of traceability applications, and the importance of links in that domain confers relation types a peculiar attention since it helps understand the rationale behind relationship - it informs not only *how* artefacts are linked but also *why* [46]. It is to note that most of the work aiming at modeling "traceability" actually models traceability **links**. This suites the OMG standard definition of traceability which consider the only input/output of transformations [87].
- An **artefact** **can be any** element of a system - e.g. unstructured documentation, code, design diagrams, test cases and suites... The nature of artefacts follows two main dimensions: **the development process they belong** to (e.g. specification, design, implementation, test), and their type (e.g. unstructured natural language, grammar-based code, model-based artefact). The **granularity of artefacts** is the level to which artefacts can be decomposed into sub parts. We call a **fragment**, the resulting product of the decomposition of an artefact. A fragment can be itself broken down into smaller parts (or sub-fragments), and so forth and so on. For example, a software requirement document could include a guideline for certification which in turn could include sub-sections and a model-level definition of their implications.
- **Trace integrity** is the degree of reliability that bares a trace. It is a measure that **includes** both the age of a trace, the volatility of artefacts targeted by the trace, and the automation level of tracing features. This indication is supported by **evidences** that can be quantitative or qualitative. For example, how long (how many versions ago) has the trace been identified in the system? Or, is has the trace been identified manually or automatically? Is there an automated co-evolution mechanism between traces and targeted artefacts? What is the level of experience of the trustee who identified it? The volatility of source and target artefacts are also factors that may influence the relevance and accuracy of a trace.
- **Pre-requirement and post-requirement** traceability **refer to, respectively, traces identified during the edification of specifications and during the implementation (design and**

- code) step of a specification [29]. The IEEE Guide for Software Requirements Specifications mentions **forward** and **backward** traceability, referring to the ability to follow traceability links from a source to a specific artefact, or the opposite from the artefact to its source, respectively [39] but, technically, the direction of traceability link (from source to target, or from target to source) does not make a difference.
- **Explicit links** refer to artefacts that explicitly link to each other in the concrete syntax. **Implicit links** show artefacts bondage at a syntactic or semantic level without explicit link (e.g., binary class and source code artefact with the same name - or the same prefix - are "linked" to each others) [58].
 - **Vertical traceability** refers to the linkage between artefacts at **different levels of abstraction** (e.g., derives, implements, inherits) whereas **horizontal traceability** refers to artefacts at the same level (e.g., uses, depends on).
 - **Time related traceability** goes along two dimensions (e.g., the evolution of (a group of) elements through successive development tasks or the evolution of artefact properties during an execution of the system.) [89].

2.3.3 TRACEABILITY METAMODEL

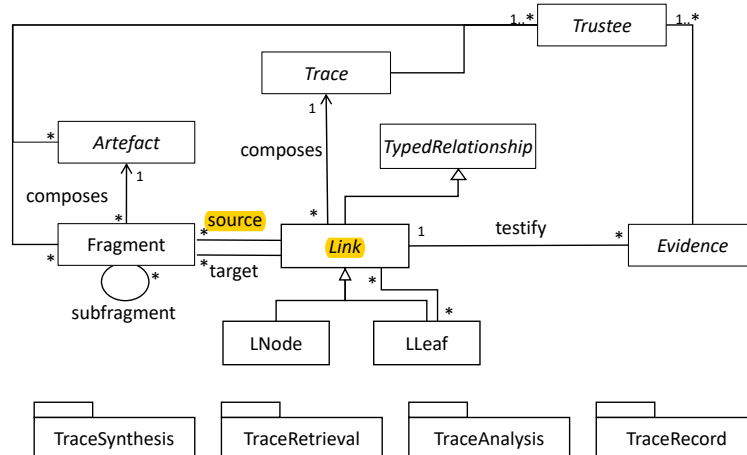


FIGURE 1: INDICATIVE METAMODEL.

As a first contribution of this work, we propose a generic traceability metamodel that helps to formalize better the relationships between the terms described above. This metamodel will also be used through out the paper to discuss and compare traceability approaches, together with the feature model presented later on. At its core, the metamodel comprises

2.4 Research questions

a generic representation of tracing artefacts: the links composing a trace, their respective evidences and trustees, and the decomposition of trace and system elements. As pictured in Fig. 1, a trace **is composed** of a set of arborescent links. Taken in its broader sense, a trace is a forest of links that connects one or more source fragment to one or more target fragment. Links are typed relationships. This typing allow for a semantic definition specific to the targeted application domain. Each link is associated with an evidence that testifies the level of confidence of that link. The reliability is affected whether the link has been created **manually or inferred automatically**, if there is a rule that assess if the link is still valid or not, if it is not deprecated, and so forth and so on. A link refers to fragments of artefacts. These fragments are decomposition of artefacts that can be decomposed as well. Since traceability is strongly advised for validation and certification of safety intensive software systems, the relation between software or trace artefact and their (human) trustee must also be considered explicitly.

2.4 RESEARCH QUESTIONS

Beyond the overall goal of analyzing the state of the art in traceability research in software/systems engineering and identifying its open research challenges, we are especially interested in the following subtopics that will be the main focus of our analysis when evaluating the papers resulting from the survey.

2.4.1 RQ1: HOW ARE TRACES **REPRESENTED?**

We are interested in knowing what languages are used to represent trace and how expressive they are. E.g. is it possible to define types of traces? And use them to link clusters of artefacts (or just individual elements)? Can we express the **confidence** we have in those traces?

2.4.2 RQ2: HOW ARE TRACES IDENTIFIED?

Do traces need to be manually created or can be derived from existing information? If so, what methods can be used for such derivation?. These are some of the topic we would like to answer in this RQ.

2.4.3 RQ3: WHAT TOOL SUPPORT IS AVAILABLE TO MANAGE AND APPLY TRACES?

Finally we want to draw a general map of the techniques that traceability approaches use to store and maintain trace artefacts of quality.

2.5 SURVEY METHOD

In this section we depict the methodology we followed to collect papers related to traceability with a peculiar scrutiny on works aiming at modeling/representing traceability information and purposes.

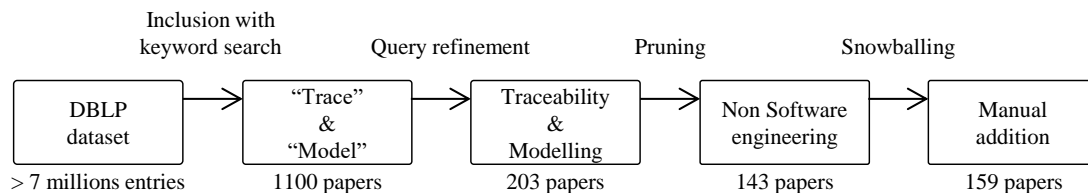


FIGURE 2: PROCESS OF THE SURVEY ON TRACEABILITY AND MODELING.

The approach selection process was conducted in three main steps illustrated in Fig. 2. First, we started selecting approaches based on our own experience about working on traceability during the last past years together with recent meta-studies on the topic, *e.g.*, [5, 18, 30, 33]. This very first selection contained a dozen distinct studies we were personally aware of.

Then, to complement our knowledge and explore the maximum of published work on traceability, we decided to mine bibliographic data sources following the literature review process established by Kitchenham and Charters [42]. Finally, we pruned the resulting selection to remove papers not related to our topic.

2.5.1 DATA SOURCE AND SEARCH STRATEGY

We used **DBLP** (2020-07-01 [2]) as our core electronic database to search for primary studies on traceability. To avoid missing possibly relevant approaches, we decided not to put a specific period constraint for the search, but we limited the scope of the search to paper of five pages or more to avoid posters, tool demos and other types of short papers. Based on the topic of this survey, we defined the terms of the search query according to the recommendations of Kitchenham and Charters [42]. As we are looking for papers proposing traceability techniques for software engineering we first considered the terms "trace" and "tracing" together with the seminal spelling "model" referring to modeling, as well as model-based variations. The idea of this first attempt was to include as many proposals as possible by using these generic model-based terms that should be part of any traceability work (as both, traces must be represented, typically as some kind of model, and many traceability approaches use models as source or target of the traces).

Nevertheless, this resulted in too large a number of results with 1100 papers matching

2.5 Survey method

the query. Potential papers, at first sight, consisted in a vast majority of false positives. Indeed, "Model" is used in too many fields, "traces" as well, and papers related to software engineering were scarce. Therefore, we refine the query including keywords more akin to modeling and software engineering, focusing more on papers mentioning traceability and not just trace and "modeling" (and its variations and related concepts: "language", "DSL", "transformation", "MDA", "MDD"). This refinement brought down the selection of 203 papers.

Here is the exact query we applied:

```
.*(([Tt]raceability|ing)|([Tt]race[rs])).* AND  
.*(([Mm]odel[- ])([Dd]riven)|([Bb]ased))|MD[DAE]|Model[l]ing|  
[Tt]ransformation|DSL|[Ll]anguage).*
```


2.5.2 PRUNING

In what follows (and as proposed by Kitchenham and Charters [42]), we describe the used inclusion and exclusion criteria. We further explain how we applied these criteria on the previous set of papers.

Inclusion criteria

1. the work is a **software** engineering approach
2. the paper is about tracing software engineering
3. traceability is the main concern

Exclusion criteria

1. the paper is  **not a primary study**
2. paper is not a white paper

Before we applied these criteria on the potential papers fetched by our query, we removed automatically papers of less than **5 pages** long. We then extracted automatically papers whose titles mention "biology", "education", "kinetics", "logistics", "physiology", "physics", "neuroscience", "agriculture", and "food" which seem to appear each in a couple of selected papers. This helped in the application of the exclusion criteria. We manually examined the 183 papers left and excluded 60 papers that did not fulfilled the criteria or were duplicates.

2.5.3 SNOWBALLING

Following the application of our methodology, we wanted to double-check that we did not miss any potentially relevant approach. Indeed, some papers could have been missed during the search process for different reasons. For instance, some workshop papers are only

indexed by ACM and some other papers have not yet been indexed by DBLP. Furthermore, some papers are not actually using the term "trace" or similar in the title (e.g., if they present so-called composition or extension approaches sometimes used as traceability mechanism). Finally, we added papers we found following the selected papers references and added papers from a specific event on traceability, the ECMFA workshop on traceability (i.e. ECMFA-TW). The final result of the overall process (including this last snowballing phase) presented 159 papers. Among them, there are 41 articles, 82 in proceedings, and 36 workshop reports (see Table 1). Fig. 3 shows the chronological distribution of this selection of publications.

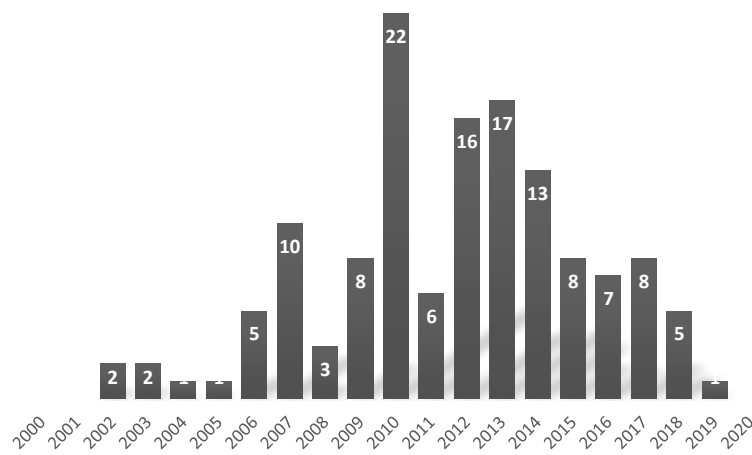


FIGURE 3: PAPERS SELECTED RELATED TO TRACEABILITY AND MODELING.

Publication type	
Journal	41
Conference	82
Workshop	36

TABLE 1: PUBLICATION TYPES OF THE SELECTED PAPERS.

2.5.4 THREADS TO VALIDITY IN THE SELECTION PROCESS

We acknowledge limitations in the execution of our survey method. First, we only used **DBLP** as a source database. Yet, it is recognized as a representative electronic database for scientific publications on software engineering and already contain more than five millions publications from more than two millions authors. Setting the limit based on the number of

2.6 Traceability, a feature model

pages alone to elude short papers is another threat to validity. Yet, it is a reproducible practice that limits the number of papers to analyse and thus helps concentrate on the topic rather than the engineering of the survey. Then, the vocabulary related to traceability is scattered among various fields of application with their respective nuances. We mitigate the risk of missing papers with a broad query and we remind the reader that, in this survey, we want to target papers explicitly mentioning traceability – and doing so, working on the emergence of a search field in itself.

2.6 TRACEABILITY, A FEATURE MODEL

Based on the analysis of the papers selected in the previous section, we proceed to answer the three research questions. For each of them, together with a more detailed explanation, we will present a feature model that summarizes the broad set of dimensions that must be considered when describing traceability proposals.

2.6.1 TRACE DEFINITION AND REPRESENTATION

As discussed in the terminology section, there is no common representation or standard of traceability. Instead, we find a plethora of applications of traceability on specific domains such as program understanding, with approaches facilitating the localisation of features [76] and bugs [43], software decomposition into relevant partitions [45] or slices [53], software reuse [82], and for supporting general explainability of the software process and product [88].

In each of these scenarios, **traces are used to support debugging** or to help gather evidences for certification. And for that, all approaches present their tailored representation for trace artefacts. The very notion of trace as well as their aim differ from one perspective to another. Researchers present generic graph-based representations particularly suitable for model-based environment [32, 73] ; while others focus on the verification and validation of software artefacts [23, 85] and traceability artefacts [67] ; and others target the maintenance of software systems with representations for change impact analysis [28] or multi-model consistency [80]. Representations are so diverse that our survey selected more than 80 papers mentioning their own distinct definitions for trace-based approaches, **with 41 metamodels effectively depicted.**

Fig. 4 shows the hierarchy of features related to the definition and the representation of trace artefacts. A peculiar focus is put on the typing of relationship. **this figure** an important feature to allow for traceability applications to specific domains in their own language. We detail as well variability in the language used and the artefacts targeted by traceability approaches and explore the needs for more quality measurement and assessment of trace artefacts.

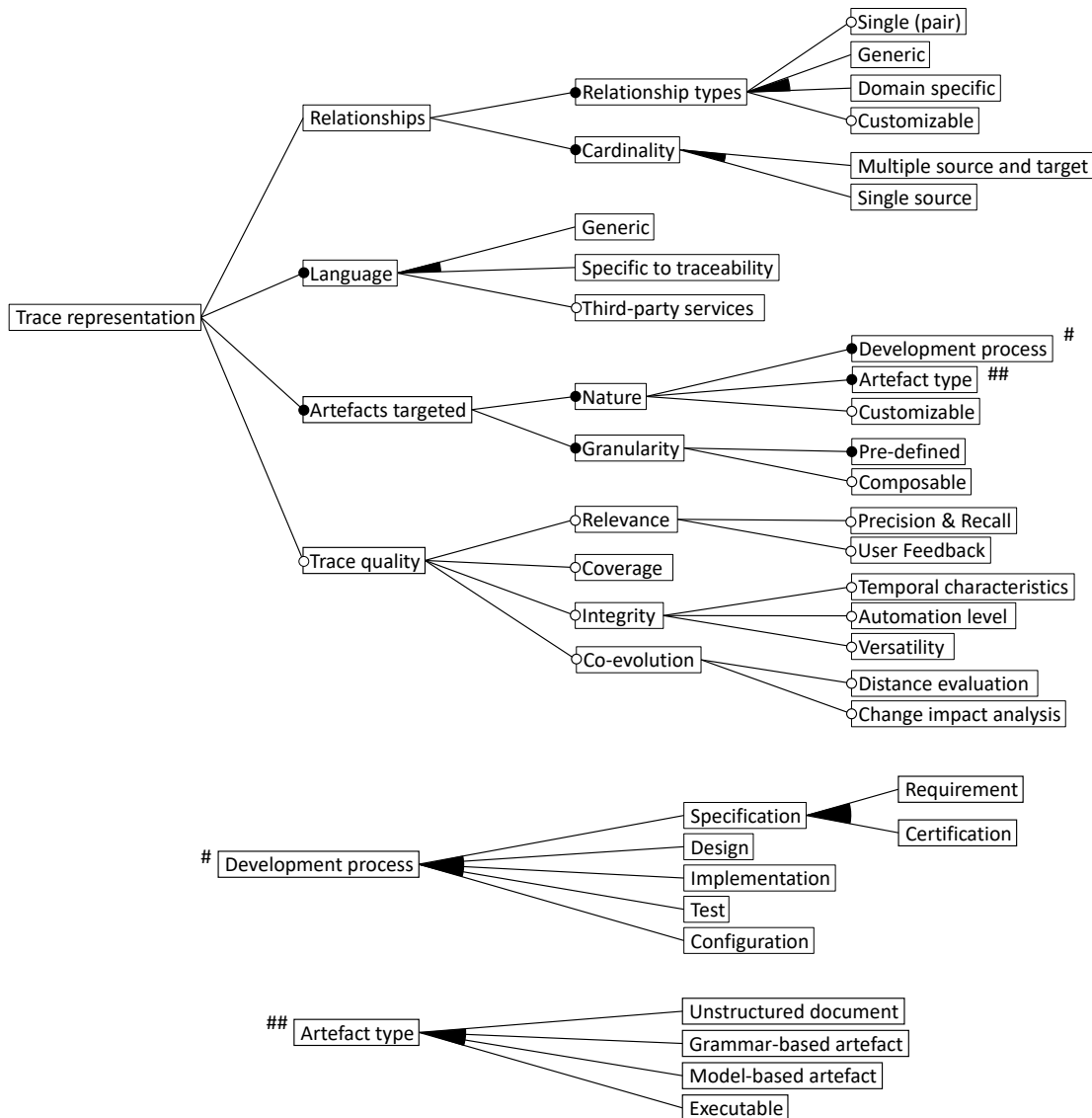


FIGURE 4: FEATURES RELATED TO THE REPRESENTATION OF A TRACE.

2.6.1.1 ARTEFACTS TARGETED

In relation to the artefacts targeted by traceability purposes we distinguish between the nature of the artefact and its granularity as both **dimensions** are important and used in the literature. For the nature, on the one hand, investigations differ on the development phase they target. Link between requirement specifications to design and code level predominate the literature with more than 50% of the papers found in the survey addressing requirement traceability. Other phases as well are targeted such as test and verification in a lesser proportion. On the other hand, the type of the artefacts is important to deduce the level of generalization transversal to the **development** lifecycle. Papers focus on four different types: unstructured document, grammar-, and model-based artefacts, and binaries. For artefacts

2.6 Traceability, a feature model

granularity, if there exist attempts to a generalizable approach were the granularity of artefacts (their level of decomposition) is customizable (e.g., by polymorphism), others only focus on a specific type to accentuate the scrutiny on specific optimizations of the execution of traceability features.

Tracing model-based development Our selection shows numerous publications referring to the application of tracing features to model-based approaches. As authors report, the use of MDE tooling such as ATL [41, 71], or the Eclipse Modeling Framework (EMF) allows the automated generation of traceability information as a side effect of executing operations [26, 87]. As a matter of fact, among the 41 metamodels selected by the survey, there are generic metamodels for end-to-end traceability [34, 36], as well as metamodels specific to engineering domain such as model transformation [3, 41, 84] or software product line [41, 84]. The Requirement Interchange Format (ReqIF) is an attempt to standardise requirement tracing in the EMF community [31]. Yet, industry has not standardised on EMF - they use a wide variety of technologies for modeling, and some of it does not conform to the usual notions of modeling technology: they may not have explicit metamodels. Paige *et al.* offer to address this issue in a call for more flexible modeling where models of different formats are associated to each other with annotations that allow automated bond or dependency inference between both application and engineering domains [60, 76].

2.6.1.2 LANGUAGE (GENERICITY)

The choice of the implementation language will impose constraints that must be evaluated beforehand. Studies presenting work related to the development of languages for traceability are legion. Some authors attempt a generic definition of traceability [7, 36], whereas others concentrate on specific uses such as SysML adaptation [53], or SPL traceability [3]. Languages specific to traceability provide the ability to represent trace artefacts with increased relevance and accuracy. Yet, they often suffer the limitation to be built *ad hoc* and lack a significant power of generalization.

We found few studies interested in the use of generic software language for traceability - even though industrial partners are interested with legacy software instrumentation for traceability and thus would benefit from traceability project coded in their legacy language [53]. **Representing traces in spreadsheets, text files, or databases offers greater freedom of expression than using a domain specific language.** This is due to the nature of actors in software development and the cognitive gap that remains between software engineers and experts of other domains. The maintenance costs turns out to grow accordingly and team members fail to update the trace artefacts [17].

2.6.1.3 RELATIONSHIP TYPES

As many authors have demonstrated, the ability offered to the user to define personalized types of relations between the artefacts of a system fosters the comprehensibility of the traces produced. We distinguish between approaches offering predefined types, most often relating to the field of software engineering (implements, inherits, uses, executes ...) and approaches allowing a domain specific typing. The formers allow increased monitoring and user-friendliness *for IT experts*. They are not very attractive to *experts in other sectors*. Allowing users to define the types of relationships specific to their area of expertise helps to fill in the cognitive gap that separates the design from the use of tracing functionalities [55] - and the communication gap that lies between respective communities [88].

SysML v2, tries to formalize traceability relationships more clearly, rather than just using a dependency-like mechanism. A emphasis is put on requirements with the definition of mechanisms in the concrete syntax to check their satisfaction [34].

Publications focusing on trace identification rarely consider other types of artefacts than the peculiar pair their optimisation target.

The literature shows also a distinction between approaches considering relationships with multiple sources and targets and relationships allowing only a single source.

2.6.1.4 TRACE QUALITY

In most of the papers we studied, quality aspects were barely mentioned. The relevance of traces, their coverage of the (part of) the system they target, their integrity or versatility have been long due for in depth investigation and standardization. The relevance of trace artefacts is mainly evaluated with precision and recall measurements while few researchers include a user feedback [18]. Their coverage is considered an important characteristics in approaches to software testing [27]. It is also used by Rath *et al.* who address the fundamental problem of missing links between commits and issues. They train a classifier with textual commit information to identify missing links between issues and commits (i.e. a lack in the coverage) [66]. Integrity of traces is addressed in work on model transformation where co-evolution figures an automatic verification of their coherence with other (versatile) software artefacts [78, 80]. The co-evolution of traces implies measuring some kind of distance between artefacts (syntactic, cognitive, geographic, cultural...) [10]. It also refers to the analysis of the changes of the system that impact traceability artefacts [28, 85]. In our survey, nine papers address artefacts co-evolution and 17 tackle model transformation limitations. These latter figuring a valuable tool to automate co-evolution tasks. The other quality concerns are left for future work and remain mainly an open issue.

A few publications relate the quality of their work on the computation of aggregated val-

2.6 Traceability, a feature model

ues, evaluated against company (or project specific) thresholds [16]. They make use of rules to automate the computation of customizable analyses and show that query, metric and rules are a powerful combination to do so. There is a need to understand clearly the purpose of trace-based approaches to define explicit measurements to evaluate traces accuracy, relevance, and deterioration.

2.6.2 TRACE IDENTIFICATION

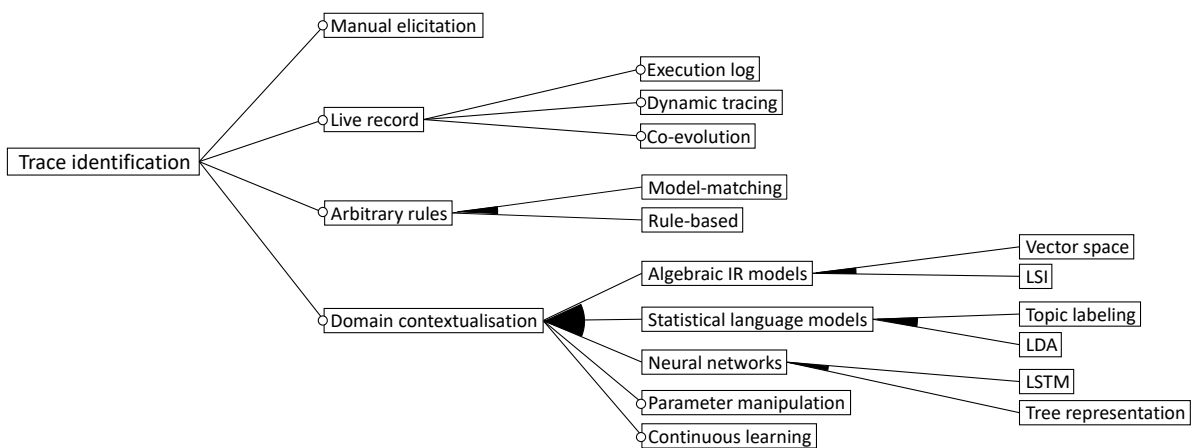


FIGURE 5: FEATURES RELATED TO THE IDENTIFICATION OF TRACE LINKS

Fig. 5 shows the hierarchy of features related to the identification of traces: the manual elicitation of traces, their live record during execution and evolution, rule-based alternatives to assist the user with automation potential, and AI-augmented identification with domain contextualization.

2.6.2.1 MANUAL ELICITATION AND RECORDING INSTRUMENTATION

Manual elicitation makes possible to create traces in an *ad hoc* manner. As an example, our industrial partner chose to hire a developer to elicit trace links necessary for a certification commitment. This was chosen rather than a (semi-)automated approach.

Teams investigate as well the live record of traces during the execution and the evolution of software artefacts. There are initiatives to instrument existing languages such as ATL with rich log generation [44, 71], while others consider trace record an aspect that can be plugged on current existing languages [64, 71]. Ziegenhagen *et al.* mix execution traces with metadatas [90], and use developer interaction records [91] to enrich existing traceability artefact.

Model transformations are considered the hearth and soul of MDE and, consequently,

numerous studies attempt to enrich trace generation during transformation execution [44, 70, 84]. This allows a semantically rich tracing of target and source artefacts [59].

2.6.2.2 ~~USE OF~~ ARBITRARY RULES

Teams identify rules that help retrieve and maintain traceability relations [52, 79]. Antoniol *et al.* use the mnemonics for identifiers to establish trace identification rules [4]. At the model level, Grammel *et al.* use a graph-based model matching technique to exploit metamodel matching techniques for the generation of trace links for arbitrary source and target models [32], and Saada *et al.* recover execution traces of model transformation using genetic algorithms [70].

2.6.2.3 DOMAIN CONTEXTUALISATION

Borillo *et al.* published a precursor article on the use of information retrieval techniques for linguistics applied to spatial software engineering. They opened the box for AI-augmented traceability [12]. Today, domain contextualization with means of machine learning for topic modeling, word embedding, and more generally knowledge extraction from unorganized text documents is the most studied feature of traceability [33, 88]. We found 22 approaches dedicated to this topic in our survey.

Researchers first extracted word vectors from natural language artefacts to take account of the neighbouring words a term in the application domain may relate to [19]. This effort made the identification of bonds between requirement specifications and other artefacts possible with a gradually improving precision. Since then, many other information retrieval techniques for natural language processing were applied with success [6]. Studies on domain contextualization are separated into three subgroups according to the type of tools used (algebraic information retrieval models, statistical language models, and neural network). For example, Florez *et al.* derivate fine grained requirement to source code links [25], Rath *et al.* complete missing links between commits and issues [66], Marcus *et al.* identify links between documentation and source code [49]. An interesting publication from Poshyvanyk *et al.* shows that mixing expertise both in information retrieval techniques and engineering domains gives far better results than expertizes taken separately (they exemplify their theory with feature localization). Teams are also using genetic algorithms to cope with the variety of algorithms and parameters these approaches use [48, 61], and structural information to foster methodologies interweaving [62]. Unfortunately, a common critique rose against these positive results. Too many teams compete with each others to accomplish better quantified precision and recall when too few attempt at qualifying the overall relation between these measurement and the effective impact on software development [18, 77]. Borg *et al.* conclude their systematic literature mapping on information retrieval approaches to traceability

2.6 Traceability, a feature model

noticing that there are no empirical evidence that any IR model outperforms another model consistently [11].

The ability to continuously improve the learning process is mentioned in the literature but we found no evidence of its application. Whereas all study tend to focus on the identification of trace links, Rosenkranz *et al.* investigate the impact of language quality for the retrieval of information [68].

2.6.3 TRACE MANAGEMENT

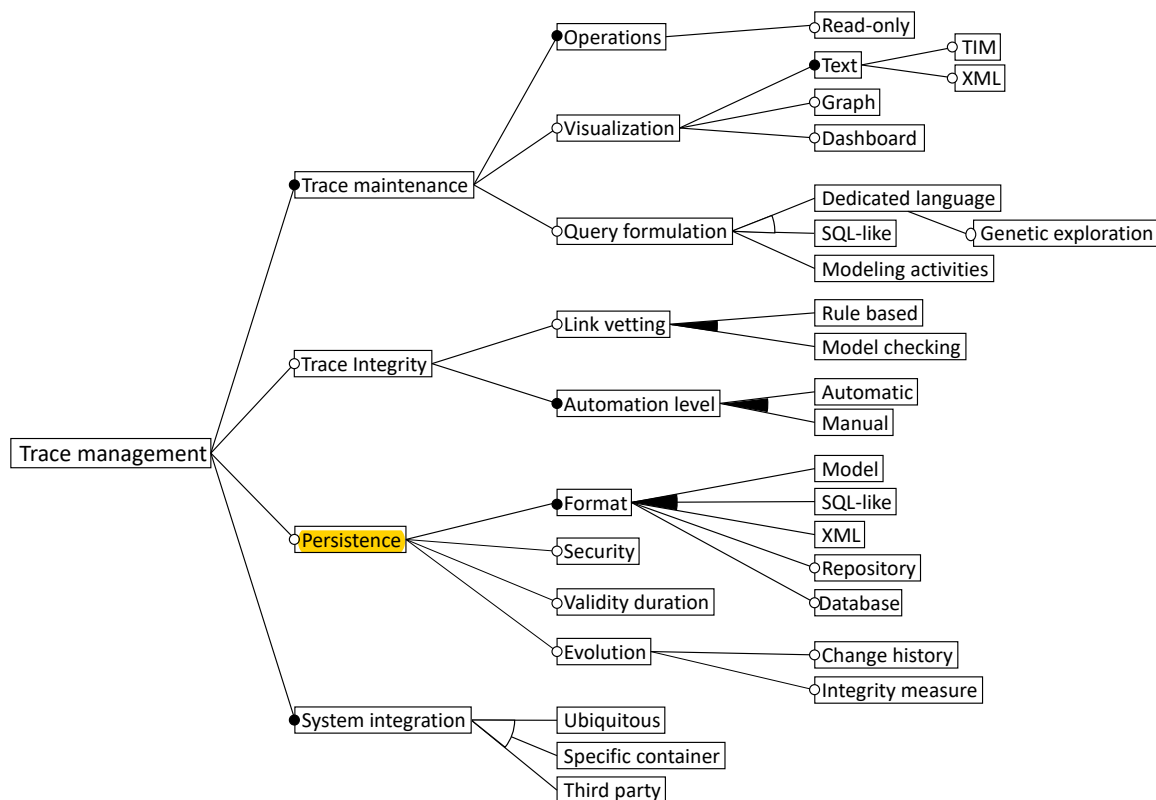


FIGURE 6: TOOL SUPPORT FOR TRACEABILITY MANAGEMENT.

Fig. 6 shows the hierarchy of features related to the management of trace artefacts. We distinguish between the actual maintenance of trace artefacts, the evaluation of their integrity, the means of persistence, and the level of integration in software systems.

2.6.3.1 TRACE MAINTENANCE

Trace links suffers a gradual decay that must be considered seriously to avoid having to re-elicit traces every time they need be analysed. A manual maintenance is out of reach

for the amount of information such inspections would involve is enormous. This motivates a serious consideration of automated ways to **visualise and retrieve** them. Co-evolution techniques [22, 52] and visualization initiatives [24] attempt to tackle the burden to maintain trace links up-to-date [16, 75].

More precisely, structured text, in the form of metamodel instances or XML sheets allows query-based mining of trace datasets. Hyper-text links is a *de facto* standard to browse trace links. Indeed, **following links branches through successive clicks has become almost natural**. The use of graphical representations stimulate human perception and the integration of such technique in traceability frameworks is a useful feature to augment tracing **awareness**. Successful dashboards offer to show simultaneously and synchronously multiple perspectives [36, 69, 72]. In parallel, allowing a rich formulation of queries to assist the exploration of existing traces will help reducing the amount of information users need to navigate through [16, 20]. Querying depends on the type of representation of traceability artefacts. SQL-like languages benefit from a long history of information mining while dedicated languages offers better legibility. Genetic programming has also permitted the automation of query formulation. Finally, model matching can be used to filter interesting traces from a repository.

All in all, assisting efficiently end-users in the retrieval, visualization and analysis of traces is not easily granted. Yet, studies on the topic remain scarce in comparison to other concerns. There is still great space for improvement and calls for more work in that direction are redundant through literature studies [5, 30].


2.6.3.2 TRACE INTEGRITY


To cope with the volatility of software products and processes, the integrity of traces must be given due consideration. Work on these questions, although called out loudly by literature studies, is scarce in practice. The first option is given with manual annotation, or vetting of trace links to inform about their level of reliability. Annotations allow a qualitative and quantitative evaluation [15]. This is the case for a back-propagation of verification and validation results between design and requirements [35]. Another approach is to impose invariant rules while manipulating traces or their targets [16]. For example, when a change occurs in an artefact targeted by a trace, if the corresponding link was identified more than two versions prior to the current version, issue an alert. Teams set up co-evolution processes to ensure the coherence of the trace is maintained [65].

2.6.3.3 TRACE PERSISTENCE

Many different storages alternatives exist for traceability artefacts. An option is to use SQL-like grammar to store and retrieve traces with the power of database tooling, or to use XML

2.7 AI Explainability as a purpose to traceability

documents to represent trace matrix in a transformable format. The industry uses a lot of informal format and link representations often remain implemented in spreadsheets, text files, databases or requirement management tools. These links deteriorate quickly during a project as time pressured team members fail to update them. Researchers aiming at a generalizable approach favour model-based representations able to express specifically defined concepts related to traceability (often in a specific domain of application) [17]. The burden of maintaining traces coherent is ased in model-based solutions.

Another concern lies in the recording of trace evolution. The trace creation should be recorded, with the successive changes that affect it. Integrity measures respective to these time stamps should be recorded as well to evaluate their evolution during a period of time. yet, we did not found any paper addressing explicitly this issue. Considering that traceability has become a major quality for certification, it is surprising to see very little interest in security concerns related to trace artefacts either.

2.6.3.4 SYSTEM INTEGRATION

Helming *et al.* use of the same modeling language for both traceability and system artefacts [37]. The conjunct use of EMF and a dedicated traceability metamodel (both written in Ecore) facilitates the integration of traceability features including graphical versions to stimulate human perception and standard analysis of traces in the native environment of the traced system.

On the other hand, Galvao *et al.* in their seminal work on traceability and MDE call for a loosely coupled traceability support that can integrate external relationship with independent representations (in another, ideally common language) [26]. To our knowledge, the most advanced research in this direction was published by Azevedo *et al.* [7].

2.7 AI EXPLAINABILITY AS A PURPOSE TO TRACEABILITY

In machine learning, the problem of reproducibility is manifold. Already because, for lack of international methodological standards, no one can verify the method by which your learning database was collected or the way in which the items were labelled. A critical question, because it is this dataset that provides the research context - or "environment" - for your algorithm: Without reliable data, an algorithm is worthless. And without an explainable algorithm, reliable data remains unsafe [9, 56]. Lead AI scientists acknowledge that AI research today suffers a frenetic pace where scientific challenges becomes competitive arguments. There is a lack of rigorous standards for empirical work as well in results dissemination and the quest for knowledge has become a run for precision and recall derivatives. Under this process, researchers do not understand why one attempt at solving a problem worked and another

failed. People implement and share techniques they do not remotely understand [74]. Initiatives prone to qualify AI processes and results, such as the ReQuest tournament, focus on efficiency and software-hardware optimization rather than explainability. Traceability is not mentioned [1]. Approaches to machine learning epistemic uncertainty do not mention it either [38].

We wondered if AI-enabled software engineering suffers from the same issues. That is, we wanted to analyse whether recent advanced on the application of AI to improve the many software engineering tasks were also oblivious to explain and trace AI-enabled architecture decisions despite a global agreement in the AI community about the importance of explainability aspects.

To make a first step toward an answer to this question, we have run a second survey. This survey focuses on papers published in a representative set conferences proposing new AI techniques for software engineering to analyse whether they include explainability and/or traceability concerns as part of their proposal.

2.7.1 SURVEY METHOD

Following the same guidelines as for the previous survey, we followed the methodology presented by Kitchenham and Charters, with the purpose to filter papers from representative software engineering international venues (namely, the International Conference on Software Engineering (ICSE), the International Conference on Automated Software Engineering (ASE), and the Symposium on the Foundations of Software Engineering (FSE)) that mention explicitly the AI nature of their work. The query we applied on the DBLP database describes terms variations between machine learning, artificial intelligence, evolutionary computation. We also included variations of the terms "intelligent" and "smart". This process output 195 papers out of 109 proceedings.

Here is the exact query we applied:

```
^.*(((Dd)eep)|([Aa]rtificial[- ] [Ii]ntelligence)|
([Nn]eural [Nn]etwork)|([Mm]achine [Ll]earning)|
[Ee]volutionary|AI|DL|[Ss]mart|[Ii]ntelligent)).*
```

We then applied the following inclusion and exclusion criteria to reach a selection of 81 papers relevant to our study. *Inclusion criteria*

1. the work is a software engineering approach
2. the work is from one of the three major software engineering venues (ICSE, FSE, or ASE)

2.8 Conclusion

3. the work mentions artificial intelligence techniques

Exclusion criteria

1. the paper is not about artificial intelligence techniques
2. the paper is not a white paper

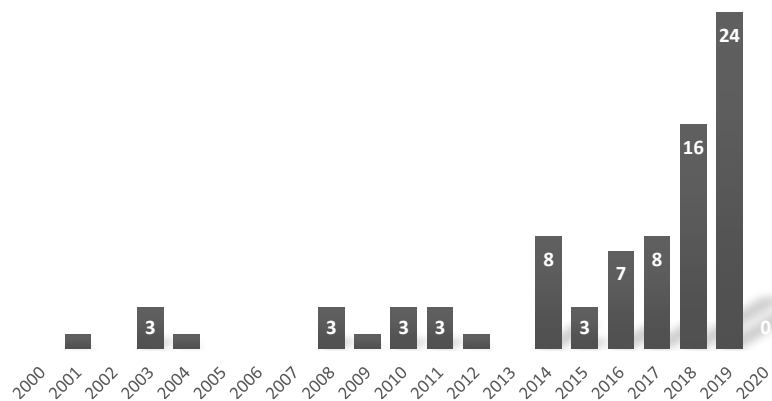


FIGURE 7: PAPERS MENTIONING ARTIFICIAL INTELLIGENCE IN SE MAJOR VENUES.

2.7.2 SURVEY ANALYSIS

We can see in Fig. 7 an exponential rise of the number of papers in the last years when. This contrasts with the gradual decrease of publications focused on traceability in the same period of time we have seen before (see Fig. 3).

But, the key question is where these 81 papers do recognize the importance traceability or related concepts as part of their proposal. This is not the case, out of the 81 AI related papers collected in the main venues for software engineering research, only two use some kind of traceability to address functional quality of AI-enabled system. Clearly, traceability, recognized as a strong weapon for verification and accountability enhancement, has not got much attention in new AI-enabled techniques for software engineering.

2.8 CONCLUSION

Our survey reveals the continuous interest in traceability even if, often, is considered as a second class citizen. Our analysis also highlights several aspects of current traceability methods that should be further developed, especially if we want to use traceability techniques to improve ongoing AI explainability initiatives.

To reach these conclusions we have conducted two surveys. The first one focused on all the works devoted to traceability methods and techniques while the second one looked at the increasing number of papers proposing AI techniques to solve software engineering problems to see if those works were integrating (or not) explainability concerns. They do not², which makes this traceability survey even more relevant.

Moreover, we argue that our feature model helps map the research area and will help understand and standardize traceability approaches and theories. This should help understand, compare and evaluate new traceability proposals. Indeed, there is a prominent call for generalization and standardization [88] in this field. Gaps between traceability-related research subfields could be better filled in with a common language bringing better communication.

We put a peculiar focus on the distinction between the modeling of traceability and the use of trace links and support our claims with two surveys. The first shows actual trends in traceability literature figuring a prominent call for generalization and standardization [88]. Current limitations are bond by technical, conceptual, and cultural gaps between research fields that could be fill in with a common language for a better communication. We argue that our feature model helps map the research area and will help understand and standardize traceability approaches and theories. The second survey shows that traceability, recognized as a strong weapon for verification and accountability enhancement, has not got much attention in AI-enabled software engineering. Instead, most work aiming to augment the automation level of traceability offer technical improvement in the automated derivation of links between artefacts of heterogeneous nature - and more specifically text artefacts [18]. We see this as a urgent call for more work on traceability for the explainability of AI.

²We see the application of AI to automatically identify new traces, e.g. [18] but not the opposite, i.e. the use of traceability to help explain AI components

REFERENCES

- [1] *ReQuEST '18: Proceedings of the 1st on Reproducible Quality-Efficient Systems Tournament on Co-Designing Pareto-Efficient Deep Learning*, New York, NY, USA, 2018. Association for Computing Machinery.
- [2] The dblp team: Monthly snapshot release of july 2020. dblp computer science bibliography., July 2020. <https://dblp.org/xml/release/dblp-2020-0701.xml.gz>.
- [3] Nicolas Anquetil, Uirá Kulesza, Ralf Mitschke, Ana Moreira, Jean-Claude Royer, Andreas Rummler, and André Sousa. A model-driven traceability framework for software product lines. *Software and Systems Modeling*, 9(4):427–451, 2010.
- [4] 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, Oct 2002.
- [5] Giuliano Antoniol, Jane Cleland-Huang, Jane Huffman Hayes, and Michael Vierhauser. Grand challenges of traceability: The next ten years. *CoRR*, abs/1710.03129, 2017.
- [6] A. Arunthavanathan, S. Shanmugathan, S. Ratnavel, V. Thiyagarajah, I. Perera, D. Meedeniya, and D. Balasubramaniam. Support for traceability management of software artefacts using natural language processing. In *2016 Moratuwa Engineering Research Conference (MERCon)*, pages 18–23, April 2016.
- [7] Bruno Azevedo. and Mario Jino. Modeling traceability in software development: A meta-model and a reference model for traceability. In *Proceedings of the 14th International Conference on Evaluation of Novel Approaches to Software Engineering - Volume 1: ENASE,,* pages 322–329. INSTICC, SciTePress, 2019.
- [8] O. Badreddin, A. Sturm, and T. C. Lethbridge. Requirement traceability: A model-based approach. In *2014 IEEE 4th International Model-Driven Requirements Engineering Workshop (MoDRE)*, pages 87–91, Aug 2014.
- [9] Andrew L. Beam, Arjun K. Manrai, and Marzyeh Ghassemi. Challenges to the Reproducibility of Machine Learning Models in Health Care. *JAMA*, 323(4):305–306, 01 2020.
- [10] Elizabeth Bjarnason, Kari Smolander, Emelie Engström, and Per Runeson. A theory of distances in software engineering. *Inf. Softw. Technol.*, 70(C):204–219, February 2016.

REFERENCES

- [11] Markus Borg, Per Runeson, and Anders Ardö. Recovering from a decade: a systematic mapping of information retrieval approaches to software traceability. *Empirical Software Engineering*, 19(6):1565–1616, 2014.
- [12] Mario Borillo, Andrée Borillo, Núria Castell, Dominique Latour, Yannick Toussaint, and M. Felisa Verdejo. Applying linguistic engineering to spatial software engineering: The traceability problem. In *Proceedings of the 10th European Conference on Artificial Intelligence*, ECAI '92, page 593–595, USA, 1992. John Wiley & Sons, Inc.
- [13] Elke Bouillon, Patrick Mäder, and Ilka Philippow. A survey on usage scenarios for requirements traceability in practice. In *Requirements Engineering: Foundation for Software Quality*, pages 158–173. Springer Berlin Heidelberg, 2013.
- [14] Pierre Bourque and Richard E. Fairley, editors. *SWEBOK: Guide to the Software Engineering Body of Knowledge*. IEEE Computer Society, Los Alamitos, CA, version 3.0 edition, 2014.
- [15] Robert Andrei Buchmann and Dimitris Karagiannis. Modelling mobile app requirements for semantic traceability. *Requirements Eng*, 22(1):41–75, jul 2015.
- [16] Hendrik Bünder, Christoph Rieger, and Herbert Kuchen. A domain-specific language for configurable traceability analysis. In *Proceedings of the 5th International Conference on Model-Driven Engineering and Software Development*. SCITEPRESS - Science and Technology Publications, 2017.
- [17] J. Cleland-Huang, B. Berenbach, S. Clark, R. Settini, and E. Romanova. Best practices for automated traceability. *Computer*, 40(6):27–35, 2007.
- [18] Jane Cleland-Huang, Orlena C. Z. Gotel, Jane Huffman Hayes, Patrick Mäder, and Andrea Zisman. Software traceability: Trends and future directions. In *Future of Software Engineering Proceedings*, FOSE 2014, page 55–69, New York, NY, USA, 2014. Association for Computing Machinery.
- [19] Andrea De Lucia, Andrian Marcus, Rocco Oliveto, and Denys Poshyvanyk. Information retrieval methods for automated traceability recovery. *Software and Systems Traceability*, pages 71–98, 2012.
- [20] T. Dietrich, J. Cleland-Huang, and Y. Shin. Learning effective query transformations for enhanced requirements trace retrieval. In *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 586–591, Nov 2013.

- [21] Nikolaos Drivalos, Dimitrios S. Kolovos, Richard F. Paige, and Kiran J. Fernandes. Engineering a dsl for software traceability. In Dragan Gašević, Ralf Lämmel, and Eric Van Wyk, editors, *Software Language Engineering*, pages 151–167, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.
- [22] Nikolaos Drivalos-Matragkas, Dimitrios S. Kolovos, Richard F. Paige, and Kiran J. Fernandes. A state-based approach to traceability maintenance. In *Proceedings of the 6th ECMFA Traceability Workshop*, ECMFA-TW '10, page 23–30, New York, NY, USA, 2010. Association for Computing Machinery.
- [23] Hubert Dubois, Marie-Agnes Peraldi-Frati, and Fadoi Lakhal. A model for requirements traceability in a heterogeneous model-based design process: Application to automotive embedded systems. In *2010 15th IEEE International Conference on Engineering of Complex Computer Systems*. IEEE, mar 2010.
- [24] F. Fittkau, J. Waller, C. Wulf, and W. Hasselbring. Live trace visualization for comprehending large software landscapes: The explorviz approach. In *2013 First IEEE Working Conference on Software Visualization (VISOFT)*, pages 1–4, Sep. 2013.
- [25] J. M. Florez. Automated fine-grained requirements-to-code traceability link recovery. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pages 222–225, May 2019.
- [26] I. Galvao and A. Goknil. Survey of traceability approaches in model-driven engineering. In *11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007)*, pages 313–313, Oct 2007.
- [27] A. Gannous and A. Andrews. Integrating safety certification into model-based testing of safety-critical systems. In *2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE)*, pages 250–260, Oct 2019.
- [28] Arda Goknil, Ivan Kurtev, Klaas [van den Berg], and Wietze Spijkerman. Change impact analysis for requirements: A metamodeling approach. *Information and Software Technology*, 56(8):950 – 972, 2014.
- [29] O. C. Z. Gotel and C. W. Finkelstein. An analysis of the requirements traceability problem. In *Proceedings of IEEE International Conference on Requirements Engineering*, pages 94–101, April 1994.
- [30] Orlena Gotel, Jane Cleland-Huang, Jane Huffman Hayes, Andrea Zisman, Alexander Egyed, Paul Grünbacher, Alex Dekhtyar, Giuliano Antoniol, Jonathan Maletic, and

REFERENCES

- Patrick Mäder. *Traceability Fundamentals*, pages 3–22. Springer London, London, 2012.
- [31] Andreas Graf, Nirmal Sasidharan, and Ömer Gürsoy. Requirements, traceability and DSLs in eclipse with the requirements interchange format (ReqIF). In *Complex Systems Design & Management*, pages 187–199. Springer Berlin Heidelberg, 2012.
- [32] Birgit Grammel, Stefan Kastenholz, and Konrad Voigt. Model matching for trace link generation in model-driven software development. In Robert B. France, Jürgen Kazmeier, Ruth Breu, and Colin Atkinson, editors, *Model Driven Engineering Languages and Systems - 15th International Conference, MODELS 2012, Innsbruck, Austria, September 30-October 5, 2012. Proceedings*, volume 7590 of *Lecture Notes in Computer Science*, pages 609–625. Springer, 2012.
- [33] Jin Guo, Jinghui Cheng, and Jane Cleland-Huang. Semantically enhanced software traceability using deep learning techniques. In *Proceedings of the 39th International Conference on Software Engineering, ICSE '17*, page 3–14. IEEE Press, 2017.
- [34] Saida Haidrar, Adil Anwar, and Ounsa Roudies. Towards a generic framework for requirements traceability management for SysML language. In *2016 4th IEEE International Colloquium on Information Science and Technology (CiSt)*. IEEE, oct 2016.
- [35] Abel Hegedus, Gabor Bergmann, Istvan Rath, and Daniel Varro. Back-annotation of simulation traces with change-driven model transformations. In *2010 8th IEEE International Conference on Software Engineering and Formal Methods*. IEEE, sep 2010.
- [36] Philipp Heisig, Jan-Philipp Steghöfer, Christopher Brink, and Sabine Sachweh. A generic traceability metamodel for enabling unified end-to-end traceability in software product lines. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing, SAC '19*, page 2344–2353, New York, NY, USA, 2019. Association for Computing Machinery.
- [37] Jonas Helming, Maximilian Koegel, Helmut Naughton, Jörn David, and Aleksandar Shterev. Traceability-based change awareness. volume 5795, pages 372–376, 10 2009.
- [38] Eyke Hüllermeier and Willem Waegeman. Aleatoric and epistemic uncertainty in machine learning: An introduction to concepts and methods, 2019.
- [39] Institute of Electrical and Electronics Engineers (IEEE). Ieee guide for software requirements specifications. *IEEE Std 830-1984*, pages 1–26, Feb 1984.

- [40] ISO. Road vehicles – Functional safety, 2011.
- [41] Álvaro Jiménez, Juan M. Vara, Verónica A. Bollati, and Esperanza Marcos. Model-driven development of model transformations supporting traces generation. In *Building Sustainable Information Systems*, pages 233–245. Springer US, 2013.
- [42] Barbara Kitchenham, O. Pearl Brereton, David Budgen, Mark Turner, John Bailey, and Stephen Linkman. Systematic literature reviews in software engineering – a systematic literature review. *Information and Software Technology*, 51(1):7 – 15, 2009. Special Section - Most Cited Articles in 2002 and Regular Research Papers.
- [43] Andrew J. Ko and Brad A. Myers. Debugging reinvented: Asking and answering why and why not questions about program behavior. In *Proceedings of the 30th International Conference on Software Engineering, ICSE '08*, page 301–310, New York, NY, USA, 2008. Association for Computing Machinery.
- [44] Thibault Béziers la Fosse, Massimo Tisi, and Jean-Marie Mottu. Injecting execution traces into a model-driven framework for program analysis. In *Software Technologies: Applications and Foundations*, pages 3–13. Springer International Publishing, 2018.
- [45] Youness Laghouaouta, Adil Anwar, Mahmoud Nassar, and Bernard Coulette. A dedicated approach for model composition traceability. *Information and Software Technology*, 91:142 – 159, 2017.
- [46] P. Mader, O. Gotel, and I. Philippow. Motivation matters in the traceability trenches. In *2009 17th IEEE International Requirements Engineering Conference*, pages 143–148, Aug 2009.
- [47] P. Maeder, I. Philippow, and M. Riebisch. A traceability link model for the unified process. In *Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD 2007)*, volume 3, pages 700–705, July 2007.
- [48] Ana Cristina Marcén, Raúl Lapeña, Oscar Pastor, and Carlos Cetina. Traceability link recovery between requirements and models using an evolutionary algorithm guided by a learning to rank algorithm: Train control and management case. *J. Syst. Softw.*, 163:110519, 2020.
- [49] A. Marcus and J. I. Maletic. Recovering documentation-to-source-code traceability links using latent semantic indexing. In *25th International Conference on Software Engineering, 2003. Proceedings.*, pages 125–135, May 2003.

REFERENCES

- [50] Jens Meinicke, Thomas Thüm, Reimar Schröter, Fabian Benduhn, Thomas Leich, and Gunter Saake. Feature traceability for feature-oriented programming. In *Mastering Software Variability with FeatureIDE*, pages 173–181. Springer International Publishing, 2017.
- [51] Yannick Moy, Emmanuel Ledinot, Hervé Delseny, Virginie Wiels, and Benjamin Monate. Testing or formal verification: Do-178c alternatives and industrial experience. *IEEE Software*, 30(3):50–57, 2013.
- [52] P. Mäder, O. Gotel, and I. Philippow. Rule-based maintenance of post-requirements traceability relations. In *2008 16th IEEE International Requirements Engineering Conference*, pages 23–32, Sep. 2008.
- [53] Shiva Nejati, Mehrdad Sabetzadeh, Davide Falessi, Lionel Briand, and Thierry Coq. A sysml-based approach to traceability management and design slicing in support of safety certification: Framework, tool support, and case studies. *Information and Software Technology*, 54(6):569 – 590, 2012. Special Section: Engineering Complex Software Systems through Multi-Agent Systems and Simulation.
- [54] Institute of Electrical and Electronics Engineers (IEEE). Ieee standard glossary of software engineering terminology. *IEEE Std 610.12-1990*, pages 1–84, Dec 1990.
- [55] Antoni Olivé. Representation of generic relationship types in conceptual modeling. In Anne Banks Pidduck, M. Tamer Ozsu, John Mylopoulos, and Carson C. Woo, editors, *Advanced Information Systems Engineering*, pages 675–691, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.
- [56] Avital Oliver, Augustus Odena, Colin Raffel, Ekin D. Cubuk, and Ian J. Goodfellow. Realistic evaluation of deep semi-supervised learning algorithms. *CoRR*, abs/1804.09170, 2018.
- [57] I. Ozkaya. What is really different in engineering ai-enabled systems? *IEEE Software*, 37(04):3–6, jul 2020.
- [58] Richard Paige, Gøran Olsen, Dimitrios Kolovos, Steffen Zschaler, and Christopher Power. Building model-driven engineering traceability classifications. 01 2010.
- [59] Richard F. Paige, Nikolaos Drivalos, Dimitrios S. Kolovos, Kiran J. Fernandes, Christopher Power, Goran K. Olsen, and Steffen Zschaler. Rigorous identification and encoding of trace-links in model-driven engineering. *Software & Systems Modeling*, 10(4):469–487, 2011.

- [60] Richard F. Paige, Athanasios Zolotas, and Dimitris Kolovos. The changing face of model-driven engineering. In *Present and Ulterior Software Engineering*, pages 103–118. Springer International Publishing, 2017.
- [61] A. Panichella, B. Dit, R. Oliveto, M. Di Penta, D. Poshynanyk, and A. De Lucia. How to effectively use topic models for software engineering tasks? an approach based on genetic algorithms. In *2013 35th International Conference on Software Engineering (ICSE)*, pages 522–531, May 2013.
- [62] A. Panichella, C. McMillan, E. Moritz, D. Palmieri, R. Oliveto, D. Poshyvanyk, and A. De Lucia. When and how using structural information to improve ir-based traceability recovery. In *2013 17th European Conference on Software Maintenance and Reengineering*, pages 199–208, March 2013.
- [63] M. C. Panis. Successful deployment of requirements traceability in a commercial engineering organization...really. In *2010 18th IEEE International Requirements Engineering Conference*, pages 303–307, Sep. 2010.
- [64] Rolf-Helge Pfeiffer, Jan Reimann, and Andrzej Wasowski. Language-independent traceability with lässig. In *Modelling Foundations and Applications*, pages 148–163. Springer International Publishing, 2014.
- [65] M. Rahimi and J. Cleland-Huang. Evolving software trace links between requirements and source code. In *2019 IEEE/ACM 10th International Symposium on Software and Systems Traceability (SST)*, pages 12–12, May 2019.
- [66] Michael Rath, Jacob Rendall, Jin L. C. Guo, Jane Cleland-Huang, and Patrick Mäder. Traceability in the wild: Automatically augmenting incomplete trace links. In *Proceedings of the 40th International Conference on Software Engineering, ICSE '18*, page 834–845, New York, NY, USA, 2018. Association for Computing Machinery.
- [67] Patrick Rempel, Patrick Mäder, Tobias Kuschke, and Jane Cleland-Huang. Mind the gap: Assessing the conformance of software traceability to relevant guidelines. In *Proceedings of the 36th International Conference on Software Engineering, ICSE 2014*, page 943–954, New York, NY, USA, 2014. Association for Computing Machinery.
- [68] Christoph Rosenkranz, Marianne Corvera Charaf, and Roland Holten. Language quality in requirements development: Tracing communication in the process of information systems development. *Journal of Information Technology*, 28(3):198–223, sep 2013.
- [69] Marcela Ruiz. *TraceME: A Traceability-Based Method for Conceptual Model Evolution - Model-Driven Techniques, Tools, Guidelines, and Open Challenges in Conceptual*

REFERENCES

- Model Evolution*, volume 312 of *Lecture Notes in Business Information Processing*. Springer, 2018.
- [70] Hajer Saada, Marianne Huchard, Clementine Nebut, and Houari Sahraoui. Recovering model transformation traces using multi-objective optimization. In *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, nov 2013.
- [71] Iván Santiago, Juan M. Vara, Valeria de Castro, and Esperanza Marcos. Measuring the effect of enabling traces generation in ATL model transformations. In *Communications in Computer and Information Science*, pages 229–240. Springer Berlin Heidelberg, 2013.
- [72] Iván Santiago, Juan Manuel Vara, María Valeria de Castro, and Esperanza Marcos. Towards the effective use of traceability in model-driven engineering projects. In Wilfred Ng, Veda C. Storey, and Juan C. Trujillo, editors, *Conceptual Modeling*, pages 429–437, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.
- [73] Hannes Schwarz, Jürgen Ebert, and Andreas Winter. Graph-based traceability: a comprehensive approach. *Software & Systems Modeling*, 9(4):473–492, 2010.
- [74] D. Sculley, Jasper Snoek, Alex Wiltschko, and Ali Rahimi. Winner’s curse? on pace, progress, and empirical rigor, 2018.
- [75] Andreas Seibel, Stefan Neumann, and Holger Giese. Dynamic hierarchical mega models: comprehensive traceability and its efficient maintenance. *Software & Systems Modeling*, 9(4):493–528, 2010.
- [76] M. Seiler, P. Hübner, and B. Paech. Comparing traceability through information retrieval, commits, interaction logs, and tags. In *2019 IEEE/ACM 10th International Symposium on Software and Systems Traceability (SST)*, pages 21–28, May 2019.
- [77] Y. Shin, J. H. Hayes, and J. Cleland-Huang. Guidelines for benchmarking automated software traceability techniques. In *2015 IEEE/ACM 8th International Symposium on Software and Systems Traceability*, pages 61–67, May 2015.
- [78] Oscar Slotosch and Mohammad Abu-Alqumsan. Modeling and safety-certification of model-based development processes. In Ina Schaefer, Dimitris Karagiannis, Andreas Vogelsang, Daniel Méndez, and Christoph Seidl, editors, *Modellierung 2018*, pages 261–273, Bonn, 2018. Gesellschaft für Informatik e.V.
- [79] George Spanoudakis, Andrea Zisman, Elena Pérez-Miñana, and Paul Krause. Rule-based generation of requirements traceability relations. *Journal of Systems and Software*, 72(2):105 – 127, 2004.

- [80] Claudia Szabo and Yufei Chen. A model-driven approach for ensuring change traceability and multi-model consistency. In *2013 22nd Australian Software Engineering Conference*. IEEE, jun 2013.
- [81] Bedir Tekinerdoğan, Christian Hofmann, Mehmet Akşit, and Jethro Bakker. Metamodel for tracing concerns across the life cycle. In Ana Moreira and John Grundy, editors, *Early Aspects: Current Challenges and Future Directions*, pages 175–194, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.
- [82] C. Tinnes, A. Biesdorf, U. Hohenstein, and F. Matthes. Ideas on improving software artifact reuse via traceability and self-awareness. In *2019 IEEE/ACM 10th International Symposium on Software and Systems Traceability (SST)*, pages 13–16, May 2019.
- [83] Tassio Vale, Eduardo Santana [de Almeida], Vander Alves, Uirá Kulesza, Nan Niu, and Ricardo [de Lima]. Software product lines traceability: A systematic mapping study. *Information and Software Technology*, 84:1 – 18, 2017.
- [84] Juan Manuel Vara, Verónica Andrea Bollati, Álvaro Jiménez, and Esperanza Marcos. Dealing with traceability in the mddof model transformations. *IEEE Trans. Software Eng.*, 40(6):555–583, 2014.
- [85] A. von Knethen. Change-oriented requirements traceability. support for evolution of embedded systems. In *International Conference on Software Maintenance, 2002. Proceedings.*, pages 482–485, Oct 2002.
- [86] Duncan J. Watts. Should social science be more solution-oriented? *Nature Human Behaviour*, 1(1):0015, 2017.
- [87] Stefan Winkler and Jens von Pilgrim. A survey of traceability in requirements engineering and model-driven development. *Software and Systems Modeling*, 9(4):529–565, 2010.
- [88] Rebekka Wohlrab, Eric Knauss, Jan-Philipp Steghöfer, Salome Maro, Anthony Anjorin, and Patrizio Pelliccione. Collaborative traceability management: a multiple case study from the perspectives of organization, process, and culture. *Requirements Engineering*, 25(1):21–45, 2020.
- [89] Y. Yu, Y. Lin, Z. Hu, S. Hidaka, H. Kato, and L. Montrieux. Maintaining invariant traceability through bidirectional transformations. In *2012 34th International Conference on Software Engineering (ICSE)*, pages 540–550, June 2012.

REFERENCES

- [90] Dennis Ziegenhagen, Andreas Speck, and Elke Pulvermueller. Expanding tracing capabilities using dynamic tracing data. In *Communications in Computer and Information Science*, pages 319–340. Springer International Publishing, 2020.
- [91] Dennis Ziegenhagen., Andreas Speck., and Elke Pulvermüller. Using developer-tool-interactions to expand tracing capabilities. In *Proceedings of the 14th International Conference on Evaluation of Novel Approaches to Software Engineering - Volume 1: ENASE*,, pages 518–525. INSTICC, SciTePress, 2019.