Empirical Studies on Software Traceability: A Mapping Study

Sofia Charalampidou[†]
Department of Mathematics and Computer Science
University of Groningen
Groningen, The Netherlands
s.charalampidou@rug.nl

Apostolos Ampatzoglou
Department of Applied Informatics
University of Macedonia
Thessaloniki, Greece
apostolos.ampatzoglou@gmail.com

Evangelos Karountzos
Department of Mathematics and Computer Science
University of Groningen
Groningen, The Netherlands
e.karountzos@gmail.com

Paris Avgeriou
Department of Mathematics and Computer Science
University of Groningen
Groningen, The Netherlands
paris@cs.rug.nl

During the last decades, software traceability has been studied in a large number of studies, from different perspectives (e.g., how to create traces? what are the benefits? etc.). This large body of knowledge needs to be better explored and exploited by both practitioners and researchers: we need an overview of different aspects of traceability and a structured way to assess and compare existing work in order to extend it with new research or apply it in practice, Thus, we have conducted a secondary study on this large corpus of primary studies, focusing on empirical studies on software traceability, without setting any further restrictions in terms of investigating a specific domain or concrete artifacts. The study explores the goals of existing approaches and the empirical methods used for their evaluation. Its main contributions are the investigation of: (a) the type of artifacts linked through traceability approaches; (b) the benefits of using artifact traceability approaches; (c) the ways of measuring their benefit; and (d) the research methods used. The results of the study suggest that: (i) requirements artifacts are dominant in traceability; (ii) the research corpus focuses on the proposal of novel techniques for establishing traceability; (iii) the main benefits are the improvement of software correctness and maintainability. Finally, although many studies include some empirical validation, there is still room for improvements, and research methods that can be used more extensively. The obtained results are discussed under the prism of both researchers and practitioners and are compared against the state-of-the-art.

1. Introduction

The term *traceability* in the software engineering domain refers to the creation of links between different types of software artifacts. There are two main ways of establishing traceability: (a) **trace recovery** which refers to after-the-fact traceability and is defined as the "approach to create trace links after the artifacts that they associate have been generated or manipulated" [1]; (b) **trace capture** which refers to real-time traceability and is defined as "the creation of trace links concurrently with the creation of the artifacts that they associate" [1]. Thus, they correspond to linking artifacts in backward and forward engineering manner respectively.

There is ample evidence to support the benefits of software traceability; here we discuss three examples. Antoniol et al. [2] investigated the use of traces between free text documentation and code either during the development or maintenance cycle. The potential benefits of such an approach include better program comprehension, easier maintenance activities, capability to assess the completeness of the product based on the traced requirements, impact analysis and a good foundation for reusing existing software. Furthermore, Alves-Foss et al. [3] suggest that traceability among artifacts enables incremental verification and validation of correctness and dependability properties, as well as analysis of compliance with existing standards and certifications during the lifespan of the project. Sundaram et al. [4], discuss two beneficial contexts of traceability, which are related to the time when traceability is performed. The first one concerns the process compliance and product improvement, when traceability is performed as part of an ongoing software development process. The second concerns software understanding and reuse, when traceability work is performed on completed project data, and its results do not contribute directly to the product improvement but rather are used in the product and process analysis.

During the last decades, software traceability has emerged as a popular topic in software engineering research, with hundreds of studies investigating numerous different aspects of traceability. Thus, secondary studies are required to provide an overview of the existing literature. Secondary studies offer several benefits; among others, they support practitioners and researchers in making informed decisions in selecting the most fitting approach for their needs, or in identifying gaps for further research respectively.

To address the need for secondary studies in this field, in this paper we report on a Systematic Mapping Study of software traceability approaches. Our study is not motivated by lack of research on software traceability; in contrast this research field is thriving and increasingly results are being applied in practice. Exactly for this reason however, researchers and practitioners need a more systematic way to assess existing work and extend it or apply it in their own context. More specifically we explore four different aspects of traceability approaches. First, we investigate the types of artifacts that are linked through traceability approaches and the corresponding development activity where traces are created; this will highlight areas for further research pertaining to artifacts that are not well supported by traceability. Second, we look into the goals of traceability approaches and how these goals are measured; this can help practitioners to find suitable approaches for their needs and researchers in adopting or calibrating measures to validate their own approaches and having them adopted in practice. Third, we examine the quality attributes in a software system that benefit from traceability; this can inform practitioners on potential benefits to system qualities from adopting traceability approaches, while researchers can gauge the effect of their own approach on qualities and how they can further improve their applicability. Fourth, we survey the research methods used (e.g., case study, experiment etc.) for validation and assessment; this can inform practitioners on the type of the provided evidence and can assist researchers in identifying areas where evidence needs to be strengthened and what type of studies are further required. Compared to other secondary studies on traceability (see Section 5), this study focuses only on empirical studies, but sets no further restrictions in terms of investigating a specific domain or concrete artifacts (more details are given in Section 5). Additionally, it is worth mentioning that this study has selected a significantly larger set of primary studies on the topic of traceability.

The choice of conducting a systematic mapping study (SMS) instead of a systematic literature review (SLR) needs further justification. According to Kitchenham et al. [5] an important criterion for selecting

the most appropriate form of a secondary study is its goal and scope. The goal of a SMS is the classification and thematic analysis of the literature on a specific topic. On the other hand, an SLR aims at "*identifying best practice with respect to specific procedures, technologies, methods or tools by aggregating information from comparative studies*" [5]. The scope of an SLR is more focused and the papers included are usually empirical studies related to a specific research question. Additionally, the information extracted from each paper is usually about individual research outcomes. As mentioned before, this study is substantially broad as it aims at classifying primary studies in terms of linked artifacts, their goals, the benefits they provide to quality attributes, and the research methods used. Therefore, a SMS is more appropriate for the goal and scope of our study. We note that we investigate only empirical studies (which is usually the practice in SLRs instead of SMSs). However, this is not uncommon; e.g., Budgen et al. [6] present a set of SMSs that report findings on empirical studies rather than the research literature as a whole.

This paper is organized as follows; In Section 2 we present the mapping study protocol we followed. In Section 3, we present the results of the mapping study and in Section 4 we discuss the results. In Section 5 we discuss related work in terms of secondary studies in the domain of software traceability, and we compare it with our own study. Finally, in Section 6 we present threats to validity and in Section 7 the conclusions of this work.

2. Study Design

This section describes the design of the proposed systematic mapping study. To design this mapping study, we followed the process proposed by Petersen et al. [7]. The essential process steps of a systematic mapping study are: (a) definition of research questions; (b) conducting the search for relevant papers; (c) screening of papers; (d) keywording of abstracts; and (e) data extraction and mapping.

2.1 Research Questions

The goal of the study is formulated according to the Goal-Question-Metrics (GQM) approach [8] as follows: "***Analyze*** *empirical studies on software traceability approaches* ***for the purpose of*** *characterization,* ***with respect to:*** *(a) the types of artifacts that are connected through traces, (b) the expected goals of using traceability and their success indicators, (c) the quality attributes that are improved through the establishment of traces, and (d) the research methods used,* ***from the point of view of*** *software engineering practitioners and researchers*". According to the abovementioned goals, and the expected contributions (see section 4) we extracted four research questions (RQ), as follows:

RQ_1: *Which types of artifacts are connected (in traced objects) and in which development activity are these artifacts created?*

With this research question we aim to investigate which types of artifacts (e.g., use cases, design artifacts, source code) are most commonly used for traceability purposes and to which artifacts they are usually linked to. In addition, we will investigate the development activity in which the artifact is developed (e.g., requirements engineering, design, implementation). In other words, we want to understand when traces are usually created (i.e., development activity) and between which artifacts. This in turn will help in identifying potential areas of traceability that have been under-studied. In

terms of development activities, we do not use a predetermined list or a specific software development lifecycle but we will report the activities as stated in the primary studies.

RQ_2: *What are the goals of the studies and how is the achievement of these goals measured?*
To answer this research question, we classify the studies according to their main goal. We look at studies aiming at *creating* traces among software artifacts, as well as those that *maintain* traces after they have been created either manually or automatically. We include both creation and maintenance as they are considered equally important in traceability management. Finally, we present success indicators that quantify the extent to which the approaches achieve their respective goal. By answering this research question, we aid practitioners in selecting the most fitting approaches for establishing and maintaining traces, and researchers in selecting pertinent measurements for validating their approaches.

RQ_3: *Which quality attributes are evaluated in the traceability studies?*
The establishment of traces is expected to have some effect on the software quality attributes. In this research question we investigate which quality attributes have been evaluated in terms of such effects, as well as the measures used for assessing the aforementioned effects. We note that the reported QAs are derived from those stated in the original studies; they were not interpreted or assumed by the authors, so as to avoid researchers' bias. The answer to this research question is expected to: (a) highlight the quality benefits that practitioners can gain by establishing and maintaining traces; and (b) explore the quality aspects that are affected by their proposed approaches.

RQ_4: *What research methods are used for validation and assessment?*
Finally, this research question will look into the most commonly used empirical research methods to validate traceability approaches or assess the effect of traces on software quality. This may provide insight on the trends of the research methods used and can indicate ideas for future work to provide stronger or different type of evidence.

2.2 Search Process

Search Scope. The search procedure of a mapping study aims at identifying as many primary studies related to the research questions as possible, using an unbiased search strategy. To achieve this goal we performed an automated search process using generic digital libraries, namely: (a) IEEE Digital Library; (b) ACM Digital Library; (c) Springer Link; and (d) Science Direct. To select the appropriate digital libraries, we adopted the inclusion criteria by Dieste et al. [9]: content update (dynamic update with new publications); availability (provide access to the full text of every research article); quality of results (test the accuracy of results returned from the query process using a small list of expected publications which are set by our team from empirical search); and versatility export (since there is a lot of noise in the retrieved results there is a process to filter the raw data). The selected sources are well-known and are constantly used in such type of studies in the software engineering field. We excluded from the list of DL indexing mechanisms, e.g., Scopus, Google Scholar, or DBLP, to: (a) avoid extensive duplication of articles; and (b) avoid the inclusion of grey literature or non-peer-reviewed materials. The search process begun at the beginning of 2017, and therefore it included studies published until the end of 2016. There was no lower boundary in the search period (i.e. we searched from the beginning of publication records).

Search Terms. In a systematic mapping study, to minimize bias and to maximize the number of sources examined, a pre-defined strategy to identify potential primary studies is required. For the automated search, the search string consisted of three main parts: *trace* **AND** *empirical* **AND** *software*. For each part, a list of alternate terms was used and connected through logical **OR** to form a more expressive query. The terms related to the "trace" part had to exist in the title of the papers, since they define the domain of interest, and we expected that in the vast majority, they should be part of the title. The terms for the "empirical" and "software" parts could exist either in their title, abstract or keywords.

Regarding the "empirical" part, the alternate terms should be concrete types of empirical research methods, since often the term "empirical" is not present in the title, abstract or keywords, resulting to missing several relevant studies. Thus, we considered several names of empirical research methods found in literature, as shown in Table 1. The first column of the table shows the research method names we considered, while the next 6 columns indicate in which sources these research methods have been considered as empirical research. To identify the list of sources, we initially referred to the most well-known papers and books dealing with empirical research (e.g., [10] and [11]). However, these sources concerned specific research methods (i.e., experiments and case studies respectively). Thus we looked for papers dealing with empirical research in a more generic point of view and came up with 5 studies providing explicitly types of empirical methods. Additionally, since ESEM is the main venue where empirical community tends to publish, we collected the research methods mentioned in their call for papers too. Similarly, we looked into the aims and scope of the *Empirical Software Engineering* journal, however no further keywords have been identified. Regarding the last column of the table, it shows how many times each keyword (research method) has been defined as empirical research in the six sources (i.e., [12]-[17]). In our search string we used as keywords only research methods which have been considered as empirical by at least two sources (green cells). We note that although the term SLR had a total sum of 2 references, we did not use the term in the search string, since we were not interested in collecting SLR studies as primary studies. Such studies would be interesting related work, but since they would not focus on the presentation or evaluation of a traceability approach they would be excluded from the list of primary studies.

Research Methods	[12]	[13]	[14]	[15]	[16]	[17]	Count
Survey	X	X	X	X	X	X	6
Case Study	X	X	X	X	X	X	6
Action Research	X	X	X	X	X	X	6
Experiment	X	X	X		X	X	5
Ethnography	X	X	X			X	4
Field Research/Study		X			X	X	3
Grounded Theory			X			X	2
Simulation			X		X		2
Quantitative Analysis					X	X	2
Experience Report / Industrial				X	X		2
SLR			X		X		2
Theoretical/Descriptive				X			1
Meta-Analysis					X		1

Qualitative		X	1
Focus Group	X		1

Table 1. Empirical research methods found in literature

Taking the above into consideration, the final search string used was the following:

("trace" OR "tracing" OR "traceability") ***AND***
("empirical" OR "case study" OR "survey" OR "experiment" OR "action research" OR "ethnography" OR "field research" OR "field study" OR "grounded theory" OR "simulation" OR "quantitative analysis" OR "experience report" OR "industrial") ***AND***
("software")

The outcomes of the automated search highly depend on the quality of the search string used, and thus several refinements were needed before the search string reached its final version, as defined above. Specifically, we iterated with different versions of the search string on a subset of the venues defined in the protocol. In each iteration, the retrieved primary studies were analyzed to verify if they are in accordance with the objective of the mapping study and the main research questions. Subsequently, improvements were applied and the process was repeated. The majority of these improvements concerned the synonyms that would be used as parts of the main components of the search terms: namely, trace AND empirical AND software. Special focus was given to the first part, in which the use of terms like "links" could lead to a large number of irrelevant results, mostly from the networks domain. All the results of this research were stored using the JabRef software, an open source bibliography reference manager. The details of the papers (i.e., title, author(s), abstract, keywords, year of publication and the name of the data source) were directly exported from the digital libraries to JabRef, using a second tool (namely: Zotero).

Primary Study Validation. To ensure high relevance of the extracted primary studies with the subject of this research, we have applied the quasi gold standard process as suggested by Zhang and Babar [18]. In particular we decided to perform a manual search in the top two journals and the two top conferences, in the domain of software engineering (TSE, TOSEM, ICSE and FSE). However, since the number of papers in these venues was considerably high, we selected to limit the manual search starting from January 2011. Next, we compared the results of the manual and automatic search indicating whether the search string was capable to return relevant studies based on the RQs. All papers forming the quasi-gold standard (4 journal and 10 conference papers in the final dataset—see Appendix B—marked with bold), were also retrieved through the automated search; therefore, the search string has been validated as appropriate.

2.3 Study Selection (Screening)

The primary studies to be selected must be relevant to empirical investigation of software traceability approaches. In line with [19], there are three phases of filtering the article set to produce the primary study data set. The first phase consists of the search process (described in Section 3.2) returning a set of candidate primary studies. This set subsequently goes through two phases of manual inspection: in the first one the inclusion/exclusion criteria are applied on each article's title, abstract/conclusions, while on the second phase they are applied on each article's full text. The inclusion/exclusion criteria that will be used in every phase are listed below:

- Inclusion criteria:

- The study introduces a software traceability approach
- The study evaluates a software traceability approach
- The full text of the study is available in English

- Exclusion criteria:
  - The study introduces an approach for tracing the same artifacts across versions (or)
  - The study concerns structural dependencies between artifacts.
  - The study introduces a traceability approach without stating explicitly the software artifacts that are linked (or)
  - The study is an editorial, position paper, keynote, opinion, tutorial, poster or panel (or)
  - The study is a previous version of a more complete paper about the same research

We note that the first exclusion criterion will lead to the exclusion of studies that trace the evolution of an artifact; therefore, the count of studies that link artifacts produced in the same development activity will be reduced (which according to the literature is very high [21]). This decision aims to exclude papers focusing on evolution analysis, which in our opinion is a large software engineering field, which deserves an investigation of its own. Every article selection phase was handled by the first two authors, who both checked all primary studies, and possible doubts were resolved by the third and fourth authors. For each data source mentioned in Section 2.2, we documented the number of papers that were returned, after removing duplicates. Also, we recorded the number of papers left for each digital library after primary study selection on the basis of title and abstract. Moreover, the number of papers finally selected from each source was recorded. The number of primary studies selected in each of the three phases is shown in Table 2 below.

Digital Library	Initial search	1st Exclusion (title, abstract, conclusions)	Final dataset
IEEE DL	431	183	**95**
ACM DL	213	94	**33**
Springer Link	31	24	**16**
Science Direct	39	19	**11**
Total	**714**	**320**	**155**

Table 2. Overview of primary studies

2.4 Keywording of Abstracts (Classification Scheme)

In [7] the authors propose keywording of abstracts as a way to develop a classification scheme, if existing schemes do not fit, and ensure that the scheme takes into account the identified primary studies. In this study, we applied keywording in order to classify artifact tracing approaches with respect to:

- The artifacts they link.
- Their benefits and the success indicators used to measure this benefit.
- Their research setting, in terms of the empirical method used and data collection methods.

Since the aforementioned information could not be obtained based only on the abstract of the study, we decided to apply the keywording technique to the articles' full texts.

2.5 Data Extraction & Mapping

During the data collection phase, we collected a set of variables from each primary study. Data collection was executed by the first two authors and possible conflicts were resolved by the third and fourth author. For every study, we extracted values to the following variables:

[V1] Author: *the list of authors*
[V2] Year: *the year of the publication*
[V3] Title: *the title of the publication*
[V4] Source: *the library where the study was found*
[V5] Venue: *the venue where the study was published*
[V6] Type of Paper: *the type of the publication (conference or workshop/ journal)*
[V7] Keywords: *the keywords reported in the publication*
[V8] Connected artifacts: *the software artifacts connected by the traceability approach*
[V9] Goal of the study: *the goal of the study in terms of research questions or expected benefits.*
[V10] Success indicators: *the means to measure the expected benefits*
[V11] Type of data analysis: *qualitative, quantitative, mixed*
[V12] Research method used: *the type of the empirical method used (case study / experiment etc.)*

Attributes [V1] – [V7] were used for Documentation reasons. All other variables were used for answering a corresponding research question. The mapping between attributes and research questions is provided in Table 3, accompanied by the analysis and presentation methods used on the data. We note that particularly for [V12], the unit of analysis is the 'empirical study'. Therefore, if one paper reports two studies explicitly, e.g., first a survey and then a case study, then this paper receives two values in [V12]. A full replication package has been made available online[1].

Research Question	Variables Used	Analysis & Presentation Method
RQ_1	[V8]	Count of connected artifacts (individual artifacts) Count of connected pairs of artifacts Grouping of identified artifacts
RQ_2	[V9] [V2]	Keywording (goals) → Count of different goals Trends (of goals per year) Cross-tabulation (goals – success indicator)
RQ_3	[V9], [V10], [V11]	Keywording (affected QAs) → Count of different affected QAs Trends (of affected QAs per year) Cross-tabulation (QAs – success indicator)

[1] http://www.cs.rug.nl/search/uploads/Resources/Charalampidou_etal_2020_MS_Traceability.zip

		Count of research methods
	[V2],	Trends (of research methods per year)
RQ_4	[V8], [V12]	Cross-tabulation (research method – pairs of artifacts RQ_1)
		Count (qualitative / quantitative)
		Cross-tabulation (research method – goal/benefit RQ_2)

Table 3. Data analysis techniques per research question

For research question **RQ_1** we investigate which are the most frequent individual artifacts, and pairs of software artifacts connected using traceability approaches. To do so we count the sets of pairs and individual artifacts found in the literature. Additionally, we group artifacts together in larger categories (e.g., group together different types of UML diagrams, or different forms of requirements).

To analyze the extracted goals of the primary studies in **RQ_2**, we apply the keywording technique. Keywording suggests the identification of keywords in the extracted data, in order to identify and map similar concepts [7]. Finally, we report the findings based on the publication year, to show how the research topics evolved over time. The same procedure has been performed for success indicators. To do so, we do a merging process [20] to group together similar success indicators. Then we performed cross-tabulation matching the identified goals to the success indicators, i.e., the ways that each goal can be measured.

For **RQ_3** we reported the results in the same way as RQ_2 by replacing goals with affected quality attributes. For **RQ_4** we count the different types of empirical research methods found in the literature. Then we show the trend throughout the years to investigate what type of research is mostly conducted as the years go by, and we relate the type of research method with the pairs of artifacts usually connected. Similarly, we perform cross-tabulation to investigate potential relations between the research methods and the goals/benefits studied. Finally, we report on the type of analysis (i.e., qualitative, quantitative or mixed).

3. Results

In this section we present the results of this study, organized by research question. Therefore, in Section 3.1 we investigate the types of artifacts that are being traced along different software development activities (RQ_1). In Section 3.2, we present the results on RQ_2, in which we studied the goals of proposing new traceability approaches. Finally, in Section 3.3, we present an overview on the impact of traceability on quality attributes, whereas in Section 3.4 we discuss the empirical setting of the validation.

3.1 Types of Traceability Artifacts and Development Activities (RQ_1)

This section aims to summarize the findings of our study with respect to the artifacts that are connected with traceability links, and the development activities in which they are produced. The sub-section is organized in two parts: the first one reports artifacts in isolation: the number of studies in which the artifacts are used regardless of the artifacts they are linked to; the second presents results on the pairs of artifacts that are being connected. Such a distinction is useful to understand both the types of artifacts that are most frequently traced (e.g., if requirements are more frequently traced than tests) as well as the pairs of artifacts that are often linked.

In Table 4, we present the top-15 most frequently investigated *individual* artifact types. For each artifact we denote the development activity in which it is produced (e.g., R: requirements engineering, D: design, I: implementation, T: testing), and the count of primary studies, in which we have identified them. We note that we had no pre-determined list of development activities; in other words, these four activities (R, D, I, T) emerged from the data. The mapping of development activities and artifacts is not a trivial task, due to the existence of various processes and Software Development Lifecycle Models (SDLC) that perform this mapping in different ways. To perform the mapping of artifacts to activities, we used a number of sources, as there was no single source that contained all the mappings we found in the data. Specifically, we used four process models (RUP, OpenUP, ICONIX, and Scrum) and the IEEE 830 Standard. For example, the "Software Architecture Document" was mapped to the design activity according to RUP (RUP calls activities as workflows), whereas the term "Use-Case Model" was mapped to the requirements activity in both OpenUP and ICONIX (OpenUP calls activities as domains while ICONIX calls them disciplines. In such cases (i.e., more than one SDLC model suggesting the same mapping) only one has been in the tables below added for simplicity. In cases of artifacts that can be mapped to more than one activity depending on the SDLC model, we map the artifact to the activity that produces it; however, we provide a reference to both SDLC models. For example, the "reported bugs/issues", which can be treated as parts of testing (in RUP) or requirements (since they are fed as backlog items in SCRUM), we map them to testing, since initially bugs are considered as an outcome of testing and later on, they are fed back to the system as requirements. We also note that the level of detail in which the artifact is being presented, depends exclusively on the reporting of the primary study. For instance in the 1st row we identified the term "Requirements (in general)" and in the 5th row the term "use case"; this is because 31 papers were explicitly referring to use cases (or provided concrete examples with use cases), whereas 70 were referring to requirements, without clarifying how they are specified (e.g., use cases, user stories, natural language, etc.).

Artifact Types	Development Activity	Count
Requirements (in general)[2]	R	76
Source Code[2]	I	60
Classes[3]	I	42
UML Design Diagrams[2]	D	33
Use Cases[2]	R	31
Test Cases[2]	T	29
Features/ Functional requirements[4]	R	24
Methods/Functions/Operations[3]	I	20
Design Models[2]	D	10
Bug Reports / Issues[3, 5]	T	9

[2] Rational Unified Process (RUP)

[3] Open Unified Process (OpenUP)

[4] 830 IEE Standard for Software Requirements Specification

Artifact Types	Development Activity	Count
Informal specifications / NL requirements[5]	R	9
Code Elements (objects / attributes)[3]	I	8
Design Artifacts[2]	D	7
Architecture Documentation / Description / Decision / Tactics[3]	D	7
Architectural Models (HL Design)[2]	D	6
Architectural Elements / Artifacts[2]	D	6

Table 4. Most frequently traced software artifact types

From the findings of Table 4 we can observe that the development activities of requirements engineering, design and implementation are almost equally represented in terms of count of types of artifacts (i.e., number of lines in Table 4). However, design artifacts appear in fewer studies compared to requirements and implementation artifacts, since they are mostly concentrated at the bottom of the table. The fact that the term "requirements" is ranked first in the list is in accordance with the literature [21]; the same goes for "source code" in the sense that it is the only software artifact that cannot be skipped in the software development lifecycle. The fact that classes are at 3rd place, denotes that most traceability studies consider object-oriented systems, whereas the explicit reference to use cases and UML diagrams suggest that there is also a preference in terms of design language and software development process. However, this could also stem from the fact that there are very few available datasets on traceability and they are reused extensively; these datasets may skew the results, as discussed further in Section 4. We note that as test cases we categorize both "test specifications" (document) and "unit tests" (code), since the two are in the majority of the cases connected by default. For example, when a requirement is linked to a test specification, the test specification will certainly be linked to the code that is used for testing. Similarly, the existence of a unit test, without a specification (even as a comment) is highly unlikely. This is in accordance to primary studies: many of them do not distinguish between the two terms.

Activity	Artifact	Count	Activity	Artifact	Count
R	Requirements (in general)	76	I	Source Code	55
	Use Cases	31		Classes	42
	Features / Functional Requirements	20		Methods / Functions	20
	Informal Specifications / NL Requirements	9		Code Elements	8
	Concerns[2,4]	5		Packages[3]	3
D	UML Design Diagrams (in general)	12	T	Test Cases	27
	Interaction Diagrams[6]	12		Bug Reports/ Issues	9

[5] Scrum - https://www.scrum.org/

[6] ICONIX

Activity	Artifact	Count	Activity	Artifact	Count
	Design Models	10		Unit Tests[5]	4
	Design Artifacts	7		Tests (in general)[2]	4
	Architectural Models (HL Design)	6		Bugs[3]	3
	Architectural Elements / Artifacts	6		Exceptions / Execution Errors[3]	3

Table 5. Most frequently traced software artifacts per development activity

To avoid researchers' bias, we preferred to list artifacts exactly in the way they are mentioned in the primary study, and not provide any interpretation at this point. For example, although one of the most common ways of specifying functional requirements is through free natural language text, there is a chance that when authors refer to functional requirements (without making explicit the type of specification), they have in mind a more structured representation like a use case or a user story. So, in Table 5 we distinguish between: *Requirements (in general)* when authors report more generally on requirements (even without explaining if their method is available on FR or NFR); *functional requirements* when authors make explicit that they refer to FR, but without making explicit the type of specification; the explicit specification type (e.g., *use case, or NL requirements*), when that is explicitly reported in the primary studies. The same strategy applies for all development activities. Subsequently in Table 5, we present the top-5 most studied artifacts per development activity by re-grouping the results of Table 4 and including artifacts with lower counts. We observe that for each development activity, the generic artifact is the dominant one (e.g., requirement, design, source code, test case).

Next, we focus on pairs of artifacts, and we present in Table 6, the top-15 most frequently connected artifacts. Similarly to Table 4, we additionally present the development activity of the artifact. We note that in Table 6, the naming convention Artifact-1 and Artifact-2 does not imply a traceability direction between the two artifacts. Therefore, both unidirectional and bidirectional relations are treated as the same, since our goal was to explore the connected artifacts and not the direction of the relation. One important observation from Table 6 is that the 6 most frequent pairs are between requirements, implementation, and testing artifacts. This observation is interesting for two reasons: (a) it makes sense as a sequence: first a requirement is specified, then the code for that requirement is written (requirements-to-code traceability is the most frequent type of trace in the literature [21], and finally the code is tested, either starting from source code, or the tested requirement; (b) testing artifacts that were rather underrepresented in Table 4, rank very high in terms of pairs of artifacts. A more detailed view of the relations between pairs of artifacts is presented in Appendix A, in a similar way to Table 6.

Artifact 1	Artifact 2	Development Activities		Count
Requirements (in general)	Source Code	R	I	21
Use Cases	Classes	R	I	16
Requirements (in general)	Classes	R	I	14
Requirements (in general)	Requirements (in general)	R	R	11
Classes	Test Cases	I	T	10

Artifact 1	Artifact 2	Development Activities		Count
Requirements (in general)	Test Cases	R	T	10
Source Code	Test Cases	I	T	7
Interaction Diagrams	Test Cases	D	T	6
Interaction Diagrams	Classes	D	I	6
Use Cases	Test Cases	R	T	6
Use Cases	Interaction Diagrams	R	D	6
Source Code	Specifications	I	-	5
Features	Source Code	R	I	5
Requirements (in general)	Methods	R	I	5
Requirements (in general)	Design Models	R	D	5

Table 6. Most Frequently Traced Pairs of Software Artifacts

3.2 Goals of Traceability Approaches and their Success Indicators (RQ_2)

This section summarizes the goals of the proposed approaches for traceability management. In Table 7, we classify the studies according to their goal, and also list their corresponding frequencies, as well as the most frequent success indicators. We note that some studies are classified under two categories: e.g., a study that introduces a novel approach can also compare it with existing ones. The analysis has led to four main categories: (a) studies that aim at proposing approaches for establishing traces, (b) studies that investigate the impact of traceability on quality attributes, (c) studies that investigate trace maintenance and evolution, (d) studies that propose benchmarks and guidelines:

- ***Establishment of Traces***: The majority of studies in Table 7 have the general goal of proposing novel traceability techniques or improving existing ones. In total 80 studies have proposed new techniques aiming to improve the accuracy of trace extraction, whereas 32 compared existing approaches to new ones or existing ones against each other. Finally, 16 papers proposed changes or the re-configuration of existing approaches for improving their accuracy.
- ***Quality of Traces:*** The second cluster of studies discusses the aspects that are important for considering the success of traces. The most common qualities (of traces per se) that are studied are the cost for establishing the traces, and the performance (usually time) for executing automated methods for trace recovery. We note that cost and performance are both measured in time but they differ in nature: cost is the time required to develop the traces manually, whereas performance refers to the time required by tools to automatically recover the traces. Also, usability of the detection approaches and the representation of traces are highlighted by the community as of significant importance. Finally, we identified two studies that deal with the scalability of trace recovery.
- ***Trace Maintenance and Evolution:*** The third and much smaller set includes studies that deal with the evolution of traces along time and how they should be managed, and the visualization of traces.
- ***Other:*** The remaining set includes 6 studies that aim at the development of benchmarks and guidelines to be used in traceability studies, as a mean for validating approaches for establishing traces.

General Goal	Focus on	Count	Success Indicators
Approaches Establishing Traceability Links	Novel Methods	80	Precision
	Comparison	32	Recall
	Improvement	16	F-measure
Quality Attribute important for Traceability	Cost	18	Time
	Performance	16	
	Usability	8	Positive Feedback
	Scalability	2	#traces, Number of Classes
Trace Maintenance	Evolution	9	#changes
	Representation / Visualization	7	User Answers
Other	Benchmarks or Guidelines	6	None

Table 7. Classification of general goals and focus of studies

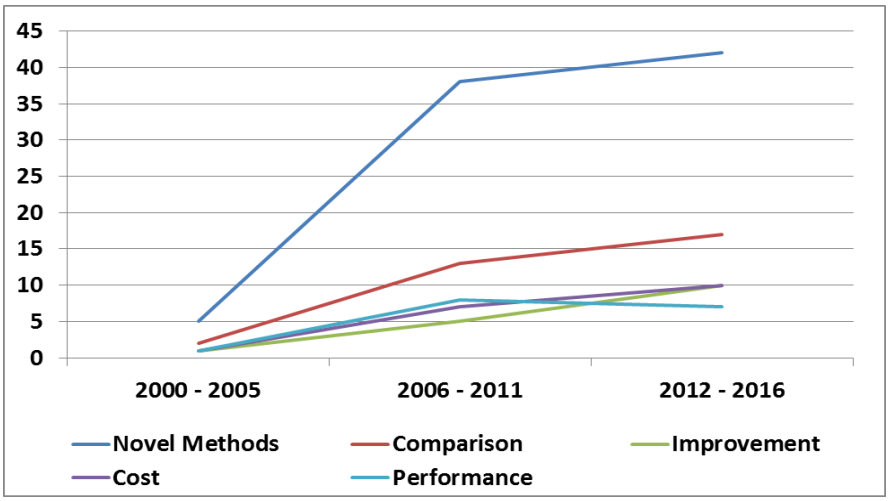

Figure 1. Chronological trends in the topics with count >10

To investigate if there are any trends in the research focus, we present in Figure 1 the chronological trends of the topics that have more than 10 studies. Regarding the proposal of new techniques, there was a large growth of papers after 2008, which later on stabilized. The rest of the groups of studies have not shown such a significant explosion in terms of number of studies, but are rather steadily increasing their numbers over the years. However, even the sharp rise of novel methods of traceability should be treated with caution, given the general increase in software engineering publications in the last decade.

3.3 Software Quality Attributes that are Evaluated in Traceability Studies (RQ_3)

Table 8 lists the top-5 quality attributes of the software that are evaluated in the primary studies, regarding how traceability affects them, listed in terms of popularity. The recorded quality attributes have been extracted from primary studies, and subsequently mapped into software quality attributes from well-

established quality models; namely: ISO-9126 [43] and ISO-25010 [42]. In Table 8 we provide a reference to the quality model where each quality attribute is defined.

Quality Characteristics	Count	Success Indicators
Modifiability [42]	15	Time to add a feature
Correctness [42]	14	#Errors / Debugging time
Analyzability [42][43]	10	Positive Users' Feedback w.r.t. Understandability
Testability [42][43]	5	LoC to be tested
Stability [43]	4	Time to perform maintenance actions

Table 8. Impact of traceability approach on software, in terms of affected Qualities

- ***Modifiability***: The relation between traceability and software modifiability (i.e., how easily a feature can be added into a system with or without traces) has been considered of great significance. For example, linking requirements to source code aids in identifying the classes (or code artifacts) that need to be updated upon a feature request.
- ***Correctness***: For the correctness, traceability links are beneficial both in terms of debugging and writing defect-free code (14 studies). For example, the existence of traces can reduce the time required to spot the code artifact that contains the bug, and therefore facilitate the easy correction of errors.
- ***Analyzability***: According to the primary studies, analyzability can refer both to the ease of understanding a piece of software that has traces, and the ease of understanding the traces themselves. For example, if a source code chunk (method or class) is linked to a requirement (which is usually more understandable, in the sense that it is expressed in natural language), then it is easier to understand, compared to non-linked artifacts.
- ***Testability***: This concerns whether software systems that have traces are easier to test (for instance, due to the existence of links between requirements and test cases). In particular, the primary studies suggest that linking artifacts to source code (ideally linking source code to test cases) reduces the amount of source code that needs to be tested for a particular requirement. In other words, the effort needed for performing the testing is reduced.
- ***Stability***: The literature suggests that whenever a trace is available, it is possible to perform accurate change impact analysis, i.e., to identify which parts are affected when another part of the software is changed (i.e., instability). Therefore, the maintainability of the software is boosted.

We acknowledge that the lack of a reference to a specific QA in a study does not automatically suggest that the investigated traceability approach does not have an effect on this QA; it was simply not studied in the corresponding primary study. We also note that Table 8 does not guarantee the existence of sufficient level of empirical evidence to support the relation between traceability and the mentioned quality attribute. Such a research question could be part of a Systematic Literature Review (in contrast to this SMS), which could evaluate the rigor and relevance of primary studies. This interesting research direction is highlighted as a tentative future work opportunity in Section 4. Regarding the chronological trends on quality attributes, in Figure 2, we can observe that in all three cases there is an increase along time.

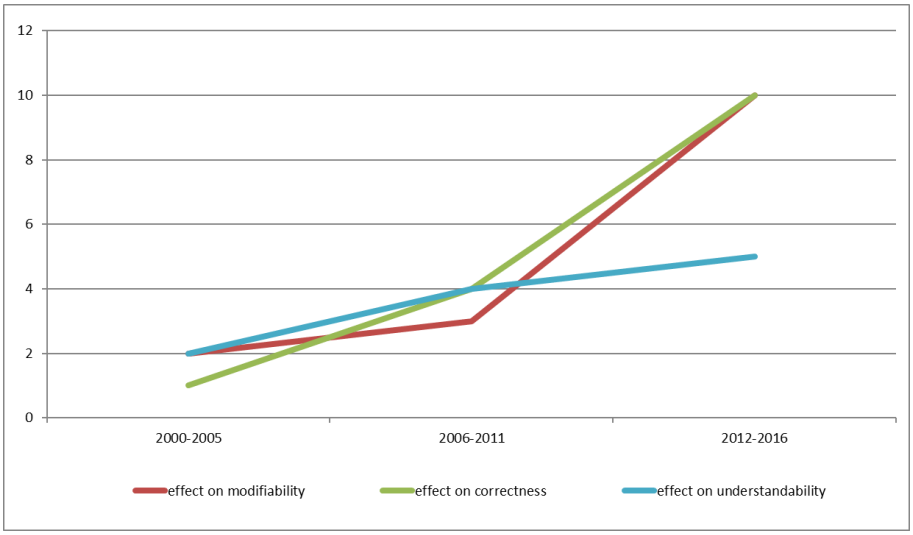

Figure 2. Chronological trends in the top 3 topics

3.4 Research Methods used (RQ_4)

In this section we present our findings regarding the research methods that have been used for validating either the methods for establishing traces, or assessing the effect of traces on quality. In Figure 3, we present some demographics of the research methods, whereas in Figure 4 the corresponding trends in time.

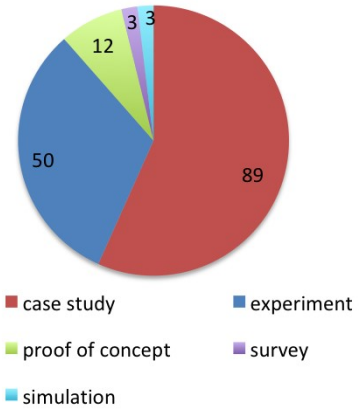

Figure 3. Count of research methods used

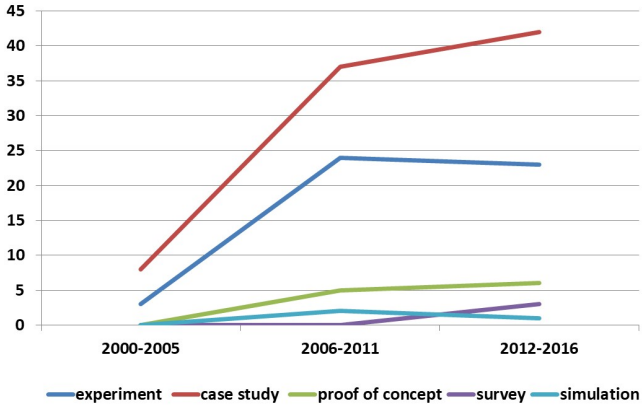

Figure 4. Chronological trends in the used research methods

The results suggest that the majority of empirical studies are based on observing the phenomenon in its current environment (case studies), followed by experiments that control the environment. An interesting observation on the corpus of empirical studies in the traceability domain is that there is a significant portion of papers (~12%) in which there is a mismatch between the actual research method used, and the one explicitly reported by the authors of the primary studies. In particular, we have identified 9 primary studies in which the authors claim that they have performed a case study, but in practice they have only performed a proof-of-concept. Similarly, in another 9 primary studies, there was a mis-classification in terms of used empirical methods (i.e., case studies instead of experiments, or vice-versa). This finding explains

a possible contradiction to related work (e.g., [37]), which suggests that experiments are the most common research method, without however performing the aforementioned crosschecking. Regarding trend analysis, we can observe that proof-of-concept studies and simulations are constant over the years. Additionally, from 2009 onwards, surveys started to appear.

By cross-tabulating the pairs of development activities in which artifacts are traced and the employed research methods, we can observe that the majority of case studies includes at least one artifact produced at the requirements engineering activity, whereas the majority of experiments includes implementation artifacts. A more detailed view on the exact artifacts being examined with different types of studies in presented in Appendix A. Regarding the methods used in order to evaluate specific empirical goals, in Table 9, we present the cross tabulation of study goals with research methods. From the results of Table 9, we can observe that to evaluate the accuracy of novel trace recovery approaches, case studies are usually performed. This result is expected since real-world projects are used as units of analysis, and the links have already been manually created and are used as the golden standard; therefore, there is no need to control other factors. Additionally, to assess the cost of using traces, experiments is the most common research method.

Research Method	Goal	Count
Case Study	Novel Methods	115
	Improvement	45
	Comparison	43
	Cost	23
	Performance	23
Experiment	Cost	92
	Evolution	84
	Novel Methods	67
	Comparison	28
	Factor	20
Proof of Concept	Usability	15
	Benefits	13
Survey	Validation	4
	Benefits	1
Simulation	Novel Methods	2
	Performance	2

Table 9. Cross tabs of study goals with research methods

Based on the data of Figure 5, we can observe that the majority of traceability studies employ quantitative analysis (72%), 16% facilitates qualitative analysis, whereas both qualitative and quantitative research methods are employed in 11% of the studies. This difference suggests that there is a need for more qualitative studies that build upon experts' opinion (through interviews, focus groups, etc.), that rely on natural

language instruments, and allow the participants of the study to express their view on the phenomenon under study. In most of the cases, the analysis has been performed with artifact analysis, and numerical data points.

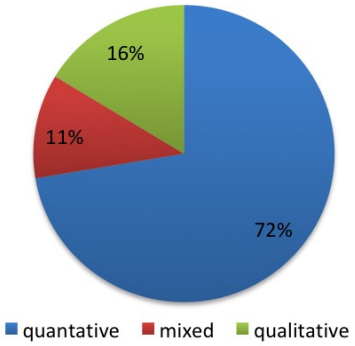

Figure 5. Frequencies on the types of data analysis methods

4. Discussion

Most Traced Artifacts. The most often traced artifacts are *software requirements*, followed by *source code*. The combination of the two artifacts is also the most commonly traced pair in the literature. One potential explanation (among others) is that these two artifacts are the most common ones used in practice: regardless of the development process used, requirements are formed in some kind (natural language, use cases, user stories, etc.) and of course source code is developed. Another tentative explanation on this could be the fact that there are very few traceability datasets publicly available (65%); and a significant portion of the studies (44%) reuse them. Therefore, to some extent this result might be related to the existence of available datasets. Also, the majority of the studies that discuss or identify benefits of traceability use these two artifacts as examples (e.g., [22], [23], [24] etc.). In addition, requirements and source code represent the problem and the solution respectively. During maintenance, both the problem and solution are constantly updated, so traces among them facilitate changing one to reflect changes on the other. In contrast, the artifacts that are not well studied, constitute potential areas for traceability research, provided of course that researchers identify practical needs to expand to such artifacts.

Another interesting observation from the data is that for each development activity, the generic artifact is the dominant one (e.g., requirements, design, source code, test cases) instead of specific artifacts (e.g., use cases, interaction diagrams, classes)—the same observation can be made by inspecting the results of other secondary studies (e.g., [37]). This leads to the conclusion that the description of current traceability approaches is not explicit, in terms of starting and ending artifacts; for example, they may aim at connecting source code to another artifact, without specifying the level of granularity (e.g., class, method, package, etc.). This more coarse-grained approach on the one hand, favors generality (i.e., the approach can be applied to all source code elements); on the other hand, it might ignore characteristics of the specific artifacts. The same finding on the coarse-grained treatment of artifacts and development activities has also been underlined by Javed and Zdun [38]. Additionally, although the approach is introduced at a coarse-grained artifact (e.g., source code level), the validation is performed only on one type of this artifact (e.g.,

classes), posing a threat to the validity of the obtained empirical evidence. We thus advise researchers to state explicitly which specific artifacts their approach applies to, and ensure their validation targets the same artifacts.

An interesting extension of this work, would be a detailed meta-analysis of the datasets used in the identified papers to investigate a possible correlation of the aforementioned findings (e.g., focus on requirements and source code, focus on OO systems, etc.) and the used available datasets. Finally, we remind that this study has not explored the directionality of the established traces. Such an investigation would be very relevant, since in most of the cases the direction is not specified in the primary studies: readers may assume that a tool can automatically traverse links in both directions, but this might not be true. Since the direction of the link is a crucial feature of any traceability approach, affecting how it can be used in practice, such an investigation would be an interesting extension of this study.

Goals of Traceability Approaches. Regarding the research goals of studies on traceability, our results suggest that the *introduction of novel traceability approaches* is the most common one in the literature. These approaches are empirically validated, usually in a quantitative manner, using precision and recall as success indicators. However, there are also many studies that only focus on *comparing existing approaches*, without proposing a novel one. Although the contribution of such studies at a first glance seems lower compared to proposing a novel approach, we find that comparative studies have merit because: (a) as the corpus of research in traceability approaches grows, it is more relevant to compare them and identify scenarios in which every approach can be used, instead of adding new ones, to an already broad set of existing approaches; and (b) in empirical software engineering research, re-evaluation of methods and tools, by independent researchers (not the authors of the original studies) is very important for reliability, repeatability, and generalizability purposes. The work on proposing novel approaches, improving existing ones, or comparing approaches, highlights the paramount importance of having a baseline, through which approaches can be objectively contrasted in terms of identification accuracy. It is thus a healthy sign, that in the last years, we have identified a decent amount of studies that propose *benchmarks and guidelines for evaluating traceability approaches*. We advise researchers to gauge their proposed approaches against such benchmarks and guidelines; we also recommend practitioners to look at such comparisons in order to select approaches that fit their needs.

The second large line of research in our results is the identification of quality attributes that are used to assess traceability approaches. On the one hand, automated approaches that aim at identifying traces after-the-fact are only considered useful in practice, if their performance is acceptable. On the other hand, approaches that involve humans in the detection or trace establishment can be very resource-intensive and this cost may prohibit their use in practice. Therefore, we advise both practitioners and researchers to consider these two parameters (performance of automated approaches and cost of manual approaches) very carefully. As far as researchers are concerned, apart from the accuracy of the proposed techniques and the benefits of using traces, they should also consider the cost and/or performance of the proposed approach, to determine the cost-benefit ratio, by considering the current practice in the case company, as a baseline. Second, practitioners should place special emphasis on these two parameters in order to make informed decisions on selecting a traceability approach.

Finally, the results of the study suggest that trace development is not the only aspect that concerns researchers, but also what happens after traces have somehow been established. The research efforts are grouped in two categories. The first has to do with the visual representation of traces, focusing especially on their usability (e.g., a GUI-based information, inside the IDE) and striving for a smooth learning curve. The second deals with the management of traces, dealing with when to update traces, where to store them, in which file format etc. This is a clear implication for researchers: they need to ensure that these two key requirements (usability and management of traces) drive the development of their tools, so as to increase the chances of industrial adoption. Additionally, practitioners can exploit this line of research to identify the most user-friendly methods and tools, as well as those that provide the most fitting trace management features to their needs. We note that regarding the goals of traceability approaches our results are not comparable to existing literature, since authors of other studies (i.e., [37] and [38]), refer to the benefits of the proposed approaches (e.g., post-requirements traceability, regulatory compliance, etc.) as goals, and not to the goals of the study.

Benefit to Quality Attributes. Our results are the first among existing secondary studies that explicitly link traceability to software qualities. In particular, in RQ_3 we have explored the potential benefits that traceability brings to the software in terms of quality attributes. As a result, most of the quality attributes that have been explored as beneficial to traceability are maintenance-related: *modifiability*, *correctness*, *instability*, *testability*, and *understandability*. This indicates that traceability, although it might increase development cost (due to the extra effort to establish the traces), can potentially provide significant benefits with respect to maintenance. The positive correlation of traceability and maintainability constitutes traceability as a very promising solution to one of the most important problems in the software engineering domain, i.e., the increasing maintenance costs. This can be a convincing argument for practitioners who consider applying traceability approaches but may not be enough to get management approval. Therefore, researchers may look into the trade-off between the costs incurred while developing traces and the financial benefits obtained during maintenance. This can be an important factor in applying traceability in practice, as the return of investment during maintenance can be a decisive factor in convincing project managers to spend resources on traceability. An interesting future research direction would be an in-depth exploration of the studies that provide empirical evidence on the relation between quality attributes and traceability, so as to: (a) investigate their rigor and relevance; and (b) synthesize evidence to reach a stronger conclusion on the effect of traceability on quality.

Empiricism in Traceability Research. The level of evidence in traceability research is quite high, in the sense that many case studies and experiments have been conducted (levels 4 to 6, according to Alves et al. [28][7]). Also, a number of proof-of-concept applications have been demonstrated, to showcase the benefits of traceability in practice. Finally, surveys have recently started to appear. One interpretation for this later start of surveys compared to the other methods may be because the motivation to use surveys as research methods relies on the existence of knowledge or experience of the subjects with the topic. Indeed, the application of traceability approaches (especially automated ones) and tools (most of them are research prototypes) in the industry was rather limited until recently [29].

[7] According to Alves et al. [28] the levels of empirical evidence that a study uses can be classified as: no evidence (0), toy examples (1), experts' opinion (2), observational (3), academic/lab (4), industrial studies (5), and industrial application (6).

The overall level of evidence is encouraging, especially as it combines quantitative results from experiments (including several causal investigations) with qualitative results from case studies (including several explorative investigations). A possible interpretation is that the level of empirical evidence tends to increase over time as studies become more rigorous; therefore, future secondary studies are expected to identify papers with higher quality evidence. Another interpretation is that our study focuses on empirical papers, since we explicitly use the empirical software engineering terms in our search string; therefore, studies that are at level 0 (no evidence) have been excluded. Research methods that are under-used for the time being, are longitudinal industrial case studies and action research. Such studies, if rigorously conducted, would significantly increase the industrial relevance of traceability research and contribute towards their further adoption in practice.

Another interesting perspective on empirical relevance, is the type and size of datasets that are used to validate new approaches. Based on our findings, only 14% of the studies use industrial datasets, while 55% use open-source and 16% use artificial ones (i.e. developed by the researchers). Among the 127 dataset that are reported in the studies (~18% of the studies have no link to a dataset) approximately 75% are available to researchers. Although there is a large number of validation datasets available (most of them originating from CoEST[8]), many of them are significantly less complex in terms of lines of code (see Table 10, in which we present the mean values of each variable for the dataset of our primary studies) than industrial datasets. Consequently, there is a need for the community to create and make available real-world datasets for validation purposes, so as to improve the external validity of future studies. An alternative could be the exploration of large-scale open datasets that resemble the size and complexity of industrial ones. Finally, we note that we have found very few papers with negative results; this is indicative of research not just on traceability but on software engineering in general.

Dataset Type	Lines of Code	#of Classes	#of Requirements
Industrial	~1Mo	~1.3K	~400
Open Source	~65K	~850	~195
Academic	~15K	~34	~400

Table 10. Size of Datasets

5. Related Work

In this section we present related work in terms of other secondary studies in the domain of software traceability, and subsequently summarize and compare this work with our study. In 2005, Spanoudakis and Zisman [29] presented a software traceability roadmap which reviewed and presented (a) different classifications of traceability relations, (b) different approaches for generating, representing, recording, and maintaining traceability relations, and (c) different ways of deploying traceability relations in the software development process. In 2007, Galvao and Goknil [30] conducted a review of the state-of-the-art on the topic of traceability approaches in MDE. The authors analyzed the primary studies with respect to five general comparison criteria: representation, mapping (i.e., traceability of model elements at different levels of abstraction), scalability, change impact analysis and tool support. In 2010, Winkler and Pilgrim

[8] http://www.coest.org/

[31] investigated traceability similarities and differences also in the domains of model driven and software engineering. Their research provided a basic description of traceability and associated topics, elaborating on how traceability can be achieved and used.

Torkar et al. [32] conducted a systematic literature review on requirements traceability. The study considers primary studies during the period 1997 to 2007 and aims at answering two main research questions, regarding: (a) the existing definitions of the requirements traceability, and (b) the existing requirements traceability techniques, their challenges and the related tools found in literature. In 2017, Tufail et al. [33] performed a Systematic Literature Review also in the area of requirements traceability aiming to identify the leading models, challenges and tools in the domain during 2010 to 2017, as well as the pros and cons of the leading requirement traceability models and tools. In the same year, Omar and Dahr conducted also a systematic literature review on the same field, aiming to present to practitioners interested into finding a suitable method for tracing requirements, the most recent (from 2008-2017) requirement traceability practices and tools available [34]. Regan et al. [35] conducted a review on traceability analyzing the motivations of the organizations for implementing traceability mostly within the regulated software safety critical domain, but also by firms outside of this field. The same year they published a second review on the topic of traceability from the perspective of the implementation of traceability in real organizations. This study aimed at presenting the barriers faced by organizations while implementing traceability. Additionally, it presented proposed solutions to these barriers, while it provided a comparison of them, for organizations operating inside and outside of the domain of critical systems [36].

Another Systematic Mapping Study was conducted by Borg et al. focused on information retrieval (IR)-based trace recovery approaches. The scope of the study is limited, focusing only on traceability approaches of natural language (NL) software artifacts, during the years 1999 to 2011. The research questions that are investigated during the study consider (a) the identification of the most frequent IR approaches for tracing NL software artifacts, (b) the types of the artifacts that are most commonly linked, and (c) the level of evidence during the evaluation of these approaches [21]. In 2013 one more review on the topic of traceability were published. Nair et al. [37] collected 70 primary studies on the domain of traceability, which had been published in the requirements engineering conference in a period of 20 years. The review investigated what traceability related aspects have been studied and by whom, what types of systems have been considered, what types of artefacts have been traced and what empirical methods have been applied. Additionally, the study reported on specific challenges on the domain that have been addressed by primary studies, and tool features that have been developed to support traceability. In 2014, Javed and Zdun conducted an SLR aiming to discover the existing traceability approaches and tools between software architecture and source code, as well as the empirical evidence for these approaches, their benefits and liabilities, their relations to software architecture understanding, and issues, barriers, and challenges of the approaches [38]. Finally, [1] conducted a roadmap for software and system traceability research, through which they present a brief view of the state of the art in traceability, the biggest challenges identified and future directions for the field. However, although this work reports on existing studies on Traceability Information Models, Automated Trace Creation and Maintenance and Traceability Economics, and provides directions based on the existing literature in the field of traceability, it is not directly comparable to our work, because it stays on a higher level of abstraction.

There are several points of differentiation between the aforementioned studies and our work. An overview of the comparison between these studies and our work is presented in Table 11. In particular, for every secondary study, we present its scope (i.e., what domain or paradigm it focuses on, what types of artifacts it investigates, which venues were searched and until which year primary studies were collected) and whether it covers the research questions of our own study (i.e., whether it studies the type of artifacts most commonly linked, the goals of traceability approaches, the benefits and the research methods used). As indicated by the table, our study has performed an analysis on the biggest dataset of primary studies compared to previous related work. Also, although our study focused only on empirical studies, there were no restrictions applied in terms of a specific domain or concrete artifacts under investigation.

	Scope					Goals			
Secondary study	**Domain / Paradigm**	**Connected artifacts**	**Search venues**	**End year**	**#primary studies**	**Traced Artifacts**	**Goal of studies**	**Quality Attributes**	**Research method**
[29]		All to all	Not mentioned	Not mentioned	Not mentioned	YES	NO	NO	NO
[30]	MDE	All to all	Not mentioned	Not mentioned	Not mentioned	NO	NO	NO	NO
[31]	MDE + Requirements		main conferences, workshops, and journals of both the RE and MDD communities + snowballing	Not mentioned	Not mentioned	NO	NO	NO	NO
[32]		Requirements to other artifacts	5 databases	2007	29	NO	NO	NO	NO
[33]	Requirements	Requirements	5 databases	2017	33	YES	NO	NO	NO
[34]	Requirements		5 databases	2015	37	NO	YES	NO	NO
[35]	(Critical Systems)		Not mentioned	Not mentioned	8	NO	NO	NO	NO
[36]	(Critical Systems) – Case Studies only		2 database + Google scholar	Not mentioned	8	NO	NO	NO	NO
[21],[41][9]	IR traceability approaches	NL artifacts to other artifacts	6 databases	2011 / 2013	79 / 25	YES	NO	NO	YES
[37]		All to all	RE Conference	2012	70	YES	YES	NO	YES
[38]		Architecture and source code	1 database + snowballing	2013	11	NO	YES	NO	YES
Our SMS	Empirical studies	All to all	4 databases	2016	156	YES	YES	YES	YES

Table 10. Comparison with related work

[9] Study [41] is a revised version (meta-analysis and synthesis) of [21], and therefore they are listed in the same row

6. Threats to Validity

In this section we present threats to validity and their mitigation, based on the guidelines provided by Ampatzoglou et al. [39]. Specifically, in Section 6.1, we report threats to validity related to study selection, in Section 6.2, we report threats related to data validity, and in Section 6.3, we report threats related to research validity.

6.1 Study Identification Validity

Study selection validity concerns the early phases of the research, i.e., the search process and the filtering of studies. In order to ensure that our search process has adequately identified all relevant studies, the primary studies that have been selected for inclusion have been carefully chosen following a well-defined protocol based on strict guidelines [40]. The identification procedure consisted of an automated search through the search engines of the most-known DLs. The search string that we used (see Section 2.2) is quite broad, since we only included the name of the investigated research method and synonyms of traceability, aiming to retrieve the maximum number of relevant studies. However, studies that adopted different terminology than the most established ones might have been excluded. The benefit of focusing only on research efforts that use standard terminology, is that we avoided using subjective criteria for characterizing the type of empirical research. To mitigate the threat of missing relevant studies, a quasi-gold standard has been used. In particular, we manually browsed the papers published in four well-established venues (namely TSE, TOSEM, FSE, and ICSE), and compared those that would be qualified for inclusion through manual extraction to those that have been retrieved automatically. Despite the fact that this process had 100% success, we need to acknowledge that using more venues for manual consideration might have yielded different results.

Next, during the article inclusion/exclusion phase, there is always a possibility of excluding relevant articles. For instance, the exclusion of studies that report on structural dependencies, or the temporal traces, might have led to excluding studies, which might be considered relevant in a wider context. To mitigate this threat, two researchers have been involved in this process, discussing any possible conflicts. On the completion of this process, a third researcher was randomly screening the selection of articles for inclusion. Also, the inclusion/exclusion criteria have been extensively discussed among the authors, so as to guarantee their clarity and prevent misinterpretations. Furthermore, from our searching space we have excluded grey literature, since the study focuses on the use of empirical evidence, which are almost never published in grey literature. As part of validation, we note that all primary studies of [21] and [32] that conform to our inclusion criteria (esp. the empirical part) have already been identified and retained in the dataset.

Additionally, although we have not identified any duplicate articles, our research protocol dictated that we check for duplicated articles, based on the abstract. Upon identification of duplicates, the most extensive one would be retained. Also, our study is not suffering from missing non-English papers and the papers published in a limited number of journals and conferences, since our search process was aiming at a large number of publication venues (including DLs as a whole) all publishing papers only in English. Moreover, we have been able to access all publications that we were interested in, since our research institutes provide us access to the used DLs.

6.2 Data Validity

Regarding data validity, the main threat is related to data extraction bias. All relevant data were extracted and recorded manually by the third author. Since this procedure is prone to some subjectivity (e.g., with respect to the mapping of artifacts to specific development activities), two researchers further inspected and refined the collected data, re-validating them. After this procedure the results were discussed among all researchers and any conflicts were resolved. One threat worth mentioning concerns the QA variable: if a QA is not mentioned in a particular study, it does not necessarily mean that this QA is not relevant to the goal of the proposed traceability approach. In most cases, authors of primary studies report findings on the QAs, which are mostly affected by their proposed approaches, instead of all affected QAs. Additionally, no publication bias is present in our results since primary studies have been collected from various venues. Thus, we argue that the obtained data points are not influenced by a small group of people.

Our secondary study is not affected by the following threats: (a) small sample size—we have been able to retrieve approx. 150 articles; (b) lack of relationships—our study was not aiming to identify any relationships among data, but only to classify and synthesize; (c) low quality of primary studies—since quality assessment is not advised for SMSs by the guidelines [7] (unless there is an explicit research question on quality assessment); and (d) selection of variables to be extracted—the straightforward research questions of our study have not raised any conflicts in the discussions among authors on which variables should be extracted. Moreover, we did not identify issues with the use of statistical analysis, in the sense that the nature of our research questions did not require hypothesis testing but only basic statistical analysis (descriptive statistics). Finally, to mitigate the researchers' bias in data interpretation and analysis the authors have discussed the data clustering for the goal of the studies, the qualities of interest, and the research methods used. However, we acknowledge that some interpretations (marked as tentative ones) are expressing the opinions of the authors, based on their understanding of the results.

6.3 Research Validity

Concerning research validity, the relevant threats concern research method bias and repeatability. Regarding the former, the authors are highly familiar with the process of conducting secondary studies, since they have been involved in a large number of secondary studies as authors and reviewers. Regarding the latter, we believe that the followed review process ensures the reliability and the safe replication of our study. First, all important decisions in our review planning have been thoroughly documented in this manuscript (see Section 2) and can be easily reproduced by other researchers. Second, the fact that the data extraction was based on the opinion of three researchers can to some extent guarantee the elimination of bias, making the dataset reliable. Third, all extracted data have been made publicly available[1], so as to enable comparison of results.

Additionally, through discussion among the authors we have set four research questions that accurately and holistically map to the set goal. This is clearly depicted by the mapping of each research question to the research sub-goals/objectives. Furthermore, in the literature we have been able to identify a substantial amount of related works that can be used for comparison to our results. In particular, for this reason we used related studies from the software engineering literature. Finally, the selection of the research method is adequate for the goal of this study and no deviations from the guidelines have been performed.

7. Conclusions

This study focuses on software traceability, i.e., the connection of software artifacts. In particular, we aim at identifying studies that provide any kind of empirical evidence related to traceability, and understanding their characteristics (in terms of linked artifacts and research methods) and goals. To achieve this goal we have performed a systematic mapping study, which has led to the inclusion of 155 studies. The results of the study suggest that requirements and source code are the most studied software artifacts, a fact that can be explained due to their nature, and importance in the software development lifecycles. Regarding the goals of the studies, our results suggest that most of the studies aim at proposing novel traceability methods, whereas the most studied quality attributes that are affected by traceability are maintainability-related. On the one hand, regarding researchers we have highlighted the following interesting research directions: (a) there is a need of a meta-analysis of the dataset in order to evaluate the level of empirical evidence; (b) traceability researchers should be more explicit in their primary studies when defining the artifacts that are being connected, and not refer to generic artifacts, such as "requirements", but rather than to concrete ones (e.g., use cases, user stories, etc.); (c) the traceability community shall expand their efforts to the connection of additional artifacts, since in the current SoTA most studies refer to requirements and code; and (d) there is a need for the development of open datasets that resemble industrial complexity. On the other hand, practitioners are encouraged to perform cost-benefit analysis for the application of traceability approaches, by considering as a benefit the high (as reported in the primary studies) maintenance gains.

Acknowledgements

We would like to appreciate our gratitude to Yikun Li for taking over additional data collection in the last review round of the paper.

References

[1] Cleland-Huang, J., Gotel, O., & Zisman, A. (2012). Software and systems traceability. Springer, London.

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

[3] Alves-Foss, J., Conte de Leon, D., & Oman, P. (2002). Experiments in the use of XML to enhance traceability between object-oriented design specifications and source code. *Proceedings of the 35th Annual Hawaii International Conference on System Sciences*, Big Island, HI, 3959-3966.

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

[5] Kitchenham, B. , Budgen, D., & Brereton, O. (2011). Using mapping studies as the basis for further research - A participant-observer case study. *Information and Software Technology*, Elsevier, 53(6), 638-651.

[6] Budgen, D., Turner, M., Brereton, P., & Kitchenham, B. (2008). Using mapping studies in software engineering, *Proceedings of the 20th Annual Workshop of the Psychology of Programming Interest Group (PPIG)*, Lancaster University, 195–204.

[7] Petersen, K., Feldt, R., Mujtaba, S., & Mattsson, M. (2008). Systematic mapping studies in software engineering. *Proceedings of the 12 th International Conference on Evaluation and Assessment in Software Engineering*, British Computer Society, 68-77.

[8] Basili, V., Caldiera, G., & Rombach, D. (1994). The Goal Question Metric Approach. *Encyclopedia of Software Engineering*, John Wiley & Sons, 528-532.

[9] Dieste, O., & Padua, A. G. (2007). Developing Search Strategies for Detecting Relevant Experiments for Systematic Reviews. *First International Symposium on Empirical Software Engineering and Measurement (ESEM)*, Madrid, 215-224.

[10] Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., & Wesslén, A. (2012). *Experimentation in Software Engineering*, Springer Publishing Company, Incorporated.

[11] Runeson, P., Host, M., Rainer, A., & Regnell, B. (2012). Case Study Research in Software Engineering: Guidelines and Examples (1st ed.). John Wiley & Sons.

[12] Easterbrook, S., Singer, J., Storey, M.A., & Damian, D. (2008). Selecting empirical methods for software engineering research. In *Guide to advanced empirical software engineering*. Springer, New York, 285–311

[13] De Magalhães, C.V.C., Da Silva, F.Q.B., & Santos, R.E.S. (2014). Investigations about replication of empirical studies in software engineering: preliminary findings from a mapping study. *In Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering (EASE '14)*. ACM, New York, NY, USA, Article 37.

[14] Hummel, M. (2014). State-of-the-Art: A Systematic Literature Review on Agile Information Systems Development. *In 47th Hawaii International Conference on System Sciences*, Waikoloa, HI, 4712-4721.

[15] Silva, F.S., Furtado Soares, F.S., Lima Peres, A., De Azevedo, I.M., Vasconcelos, A.P.L.F., Kamei, F. K., & De Lemos Meira, S. R. (2015). Using CMMI together with agile software development: A systematic review. *Information and Software Technology*, Volume 58, 20-43.

[16] International Symposium on Empirical Software Engineering and Measurement (ESEM), http://esem-conferences.org.

[17] Stol, K, Babar, J. M.A., Russo, B., & Fitzgerald, B. (2009). The use of empirical methods in Open Source Software research: Facts, trends and future directions. *In Proceedings of the 2009 ICSE Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development (FLOSS)*, IEEE Computer Society, Washington, DC, USA, 19-24.

[18] Zhang, H., and Babar, M. A. (2010). On searching relevant studies in software engineering. *In Proceedings of the 14th international conference on Evaluation and Assessment in Software Engineering (EASE)*, British Computer Society, Swinton, UK, UK, 111-120.

[19] Dybå, T., & Dingsøyr, T. (2008). Empirical studies of agile software development: A systematic review. *Information and Software Technology*, Elsevier, 50 (9-10), 833-859.

[20] Bafandeh Mayvan, B., Rasoolzadegan, A., & Ghavidel Yazdi, Z. (2017). The state of the art on design patterns. *Journal of Systems and Software*. 125, C, 93-118.

[21] Borg, M., Runeson, P., & Ardö, A. (2013). Recovering from a decade: a systematic mapping of information, retrieval approaches to software traceability. *Empirical Software Engineering*, Springer.

[22] Maia, M. A., & Lafeta, R. F. (2013). On the impact of trace-based feature location in the performance of software maintainers. *Journal of Systems and Software*, 86(4), 1023-1037.

[23] Ali, N., Sharafi, Z., Guéhéneuc, YG., Antoniol, G. (2015). An empirical study on the importance of source code entities for requirements traceability. *Empirical Software Engineering*, 20(2), 442–478.

[24] Mäder, P., & Egyed, A. (2015). Do developers benefit from requirements traceability when evolving and maintaining a software system? *Empirical Software Engineering,* 20(2), 413-441.

[25] Van Vliet, H. (2008). Software Engineering: Principles and Practice, 3rd edition, John Wiley & Sons.

[26] Galorath, D. D. (2008). Software total ownership costs: development is only job one. *Software Tech News*, 11(3).

[27] Arvanitou, E. M., Ampatzoglou, A., Chatzigeorgiou, A., & Avgeriou, P. (2015). Introducing a Ripple Effect Measure: A Theoretical and Empirical Validation. *In 9th International Symposium on Empirical Software Engineering and Measurement (ESEM)*, IEEE, 1-10.

[28] Alves, V., Niu, N., Alves, C., & Valença, G. (2010). Requirements engineering for software product lines: A systematic literature review. *Information and Software Technology*, 52(8), 806-820.

[29] Spanoudakis, G., & Zisman, A. (2005). Software traceability: a roadmap. *In Handbook of Software Engineering and Knowledge Engineering*, vol. 3--Recent Advances,World Scientific, Singapore, 395-428.

[30] Galvao, I., & Goknil, A. (2007). Survey of Traceability Approaches in Model-Driven Engineering. *In Proceedings of the 11th IEEE International Enterprise Distributed Object Computing Conference (EDOC)*, IEEE Computer Society, Washington, DC, USA, 313-313.

[31] Winkler, S., & Pilgrim, J. (2010). A survey of traceability in requirements engineering and model-driven development, *Softw. Syst. Model.* 9(4), 529-565.

[32] Torkar, R., Gorschek, T. , Feldt, R., Svahnberg, M., Akbarraja, U., Kamran, K. (2012). Requirements Traceability: A Systematic review and Industry Case Study. *International Journal of Software Engineering and Knowledge Engineering*, World Scientific Publishing, 22(03), 385-433.

[33] Tufail, H., Masood, M. F., Zeb, B., Azam, F., & Anwar, M.W. (2017). A systematic review of requirement traceability techniques and tools. *In the 2nd International Conference on System Reliability and Safety (ICSRS)*, Milan, 450-454.

[34] Omar, M., & Dahr. J. M. (2017). A Systematic Literature Review of Traceability Practices for Managing Software Requirements. *Journal of Engineering and Applied Sciences*, 12: 6870-6877.

[35] Regan, G., McCaffery, F., McDaid, K., & Flood, D. (2012a). Traceability- Why Do It? *International Conference on Software Process Improvement and Capability Determination (SPICE'12)*, 161-172

[36] Regan, G., McCaffery, F., McDaid, K., & Flood, D. (2012b). The Barriers to Traceability and their Potential Solutions: Towards a Reference Framework. *In 38th Euromicro Conference on Software Engineering and Advanced Applications*, Cesme, Izmir, 319-322.

[37] Nair, S., De la Vara, J. L., & Sen, S. (2013). A review of traceability research at the requirements engineering conference. *In 21st IEEE International Requirements Engineering Conference (RE)*, Rio de Janeiro, 222-229.

[38] Javed, A., & Zdun, U. (2014). A systematic literature review of traceability approaches between software architecture and source code. *In Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering (EASE)*, ACM, New York, NY, USA, Article 16.

[39] Ampatzoglou, A., Bibi, S., Avgeriou, P., Verbeek, M., & Chatzigeorgiou, A. (2019). Identifying, Categorizing and Mitigating Threats to Validity in Software Engineering Secondary Studies. *Information and Software Technology*, 106.

[40] Kitchenham, B., & Charters, S. (2007). Guidelines for performing systematic literature reviews in software engineering. *Technical Report EBSE 2007-001*, Keele University and Durham University.

[41] Borg M. and Runeson P., "IR in Software Traceability: From a Bird's Eye View," *2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*, Baltimore, MD, 2013, pp. 243-246

[42] ISO/IEC 25010:2011, Systems and software engineering—Systems and software Quality Requirements and Evaluation (SQuaRE)—System and software quality models, Geneva, Switzerland, 2011.

[43] ISO/IEC 9126-1:2001, Software engineering - Product quality (Part 1: Quality model), Geneva, Switzerland, 2001.