

## SURVEY

# A Systematic Literature Review of Issue-Based Requirement Traceability

YIJING LYU<sup>1</sup>, HEETAE CHO<sup>1</sup>, PILSU JUNG<sup>1,2</sup>, AND SEONAH LEE<sup>1,2</sup>, (Member, IEEE)

<sup>1</sup>Department of AI Convergence Engineering, Gyeongsang National University, Jinju-si, Gyeongsangnam-do 52828, South Korea

<sup>2</sup>Department of Aerospace and Software Engineering, Gyeongsang National University, Jinju-si, Gyeongsangnam-do 52828, South Korea

Corresponding author: Seonah Lee (saleese@gnu.ac.kr)


This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2021R1A2C1094167).

**ABSTRACT** Issue reports are software artifacts that often specify the changed requirements of software systems. As software systems evolve according to these changed requirements, issue reports have become the essential artifacts that should be covered by requirement traceability. While researchers have developed automatic approaches for establishing the traceability links of issue reports, no papers have surveyed these approaches. In this paper, we conduct a systematic literature review of issue-based requirement traceability. We searched for articles published in renowned conferences and journals in the software engineering field from 2011 to 2022. From 1,347 initial articles, we identified 40 relevant articles. We investigated four aspects of issue-based traceability: problems, artifact pairs, techniques, and evaluation targets. Our findings are as follows. First, the challenges of issue-based requirement traceability are relevant to accuracy, effort, support, information, and trustworthiness. Second, issue reports are linked to commits, source code, user reviews, and test cases. Third, the studies mainly adopted machine learning and information retrieval techniques to generate and recover trace links. Finally, the main evaluation targets were open-source projects, but open datasets were also provided.

**INDEX TERMS** Software issue reports, software requirement traceability, systematic literature review, traceability links.

## I. INTRODUCTION

Requirement traceability refers to the ability to trace relationships from software artifacts to other artifacts [1]. Requirement traceability can effectively help developers explore software artifacts or analyze the impact of a change. Modern software projects mainly adopt Issue Tracking Systems (ITSs), such as Jira, to rapidly reflect change requests in their software products. In the ITS, users can specify their requests as issue reports. Therefore, issue reports often include change requests and are considered software artifacts. In contrast to traditional requirement traceability, we named the traceability study that categorized issue reports as primary software artifacts the Issue-based Requirement Traceability (I-RT) study.

The associate editor coordinating the review of this manuscript and approving it for publication was Porfirio Tramontana .

Traditional requirement traceability studies did not handle issue reports as software artifacts but handled requirements as primary artifacts. For example, studies [2], [3], [4] generated trace links between requirements and other artifacts, such as use cases, source code, and architecture descriptions. Studies [5], [6], [7] recovered trace links between requirements and source code. Several literature surveys have even been conducted on requirement traceability [8], [9], [10], [11], [12], [13], [14], [15]. However, the surveys do not cover the studies that handle issue reports. This situation creates a gap between traditional requirement traceability and modern development practices that use ITSs. To create traceability in modern development practices, issue reports should be covered by a requirement traceability approach. Several studies have addressed requirement traceability with issue reports as part of software artifacts [16], [17], [18]. Nevertheless, to the best of our knowledge,

no comprehensive literature surveys have been conducted on I-RT.

In this paper, we conducted a Systematic Literature Review (SLR) of I-RT. First, we defined four research questions that examine four aspects of I-RT: problems, artifact pairs, techniques, and evaluation targets. Second, we searched I-RT studies published from 2011 to 2022 and found 1,347 articles from well-known conferences and journals in the software engineering field. Third, we identified 40 studies that were relevant to I-RT. Finally, we extracted and analyzed relevant information for each research question.

The remainder of this paper is organized as follows: Section II discusses literature surveys on requirement traceability. Section III describes the procedure of the conducted SLR regarding I-RT. Section IV reports the results obtained for each research question. Section V discusses the future directions of I-RT. Section VI presents the threats to the validity of this research. Section VII concludes this paper.

## II. RELATED WORK

There are two ways to conduct literature surveys: Systematic Mapping Studies (SMSs) and SLRs. An SMS categorizes and analyzes existing works and obtains a systematic map that describes a classified portfolio of the research results relevant to a particular research topic [19]. An SLR collects and analyzes existing studies and obtains supporting evidence to answer research questions [20]. The aims of SLRs and SMSs differ in that an SLR is an in-depth study of a narrow area that is completed by using specific and pointed research questions to create new knowledge through a meta-analysis of existing knowledge published in the literature, while an SMS aims to create a map of a wide research field [21].

The first group of works conducted SMSs [8], [9], [10]. Borg et al. [8] focused on Information Retrieval (IR)-based trace recovery and collected studies published from 1999 to 2011. They found that the inconsistent use of IR terminology was the main problem with the IR models used for trace recovery. Vale et al. [9] surveyed studies on traceability for software product lines and collected studies published from 2001 to 2015. They found that most studies focused on the trace links between assets at different levels of abstraction. Charalampidou et al. [10] focused on the relationships of software artifacts in traceability approaches and surveyed studies published before 2016. They found that requirements and source code are the most studied software artifacts and that the most studied quality attribute with respect to traceability is maintainability. The studies of Borg et al. [8] and Vale et al. [9] differ from our literature survey in that they focused on information retrieval techniques for traceability and software product line traceability, respectively. In contrast, we focused on the techniques used in I-RT. Charalampidou et al.'s study in [10] is partially similar to ours in that they investigated software artifacts. However, they did not include issue reports as software artifacts.

The second group conducted SLRs on requirement traceability [11], [12], [13], [14], [15]. Cleland-Huang et al. [15] conducted an SLR to review the state-of-the-art and discuss the future directions of software traceability. They collected papers published from 2003 to 2013. Mustafa and Labiche [11] conducted an SLR to model traceability among artifacts obtained from different domains of expertise. They collected papers published from 2000 to 2016. They found that few studies focused on heterogeneous artifacts, traceability tools, and precise semantics for trace links. Tufail et al. [12] focused on analyzing the models and tools used in requirement traceability studies and collected papers published from 2010 to 2017. They identified seven requirement traceability models, ten requirement traceability challenges, and fourteen requirement traceability tools. Wang et al. [13] comprehensively studied the technologies and challenges of traceability based on papers published from 2006 to 2016. They identified challenges in terms of traceability and technologies mapped to these challenges. Aung et al. [14] focused on change impact analysis and surveyed the available approaches for automatically recovering traceability links. They identified the approaches proposed from 2012 to 2019 and analyzed the research gaps between the current states of the proposed approaches and the desired states of these approaches. Tian et al. [22] surveyed studies on traceability for software maintenance and evolution and collected studies published from 2000 to 2020. They found that the two main challenges that hinder practitioners from employing the traceability practices are the quality of traceability links and the performance of traceability approaches and tools. None of the studies included issue reports as software artifacts.

## III. SYSTEMATIC LITERATURE REVIEW PROCEDURE

To understand the newly emerging theme, I-RT, we conducted an SLR by following a guideline [23]. Figure 1 presents the overall procedure of our SLR. The procedure consists of three phases: planning, searching, and analysis. In the planning phase, we defined Research Questions (RQs). In the searching phase, we formed queries, identified data sources, applied inclusion and exclusion criteria to the candidate papers, checked full text, and did snowballing. In the analysis phase, we assessed the quality of these studies and extracted information.

### A. DEFINING THE RQs

We derived RQs by considering our research purpose to provide comprehensive knowledge for the research trend of I-RT studies as follows:

- RQ1. What problems are targeted in I-RT studies?
- RQ2. Which software artifact pairs are linked to issues in I-RT studies?
- RQ3. What techniques are used in I-RT studies?
- RQ4. What evaluation targets are used in I-RT studies?

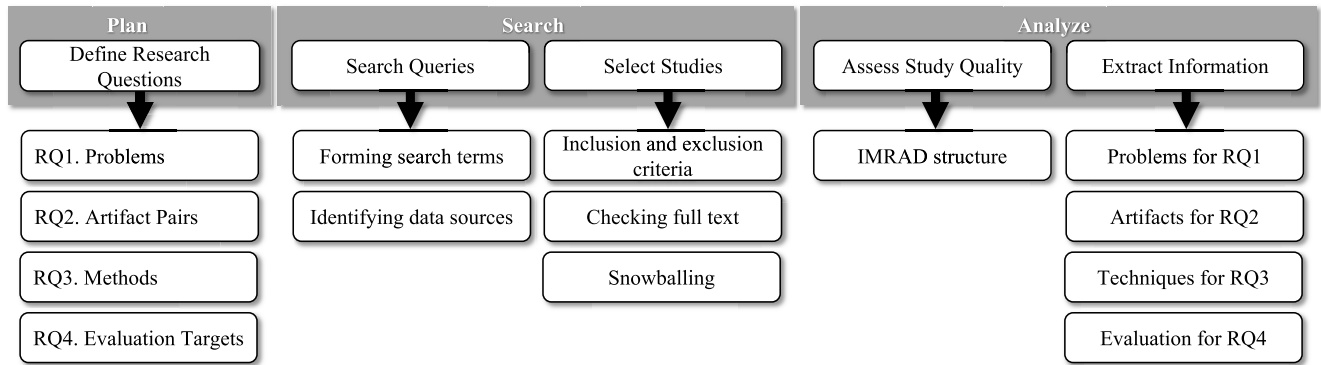


FIGURE 1. Process of systematic literature review.

**B. SEARCHING THE RELEVANT STUDIES**

Our selection process consists of five steps. First, we formed queries. Second, we then searched digital libraries with the queries. Third, we applied inclusion and exclusion criteria to filter out papers. Fourth, we examined the full texts of the remaining papers to identify the papers relevant to our research goal. Last, we performed forward/backward snowballing with the identified papers. We explain the details of each step as follows:

1) SEARCH TERMS

We used the PICO (Population, Intervention, Comparison, and Outcome) criteria to define the search terms based on the SLR guideline [23].

- *Population:* The population in this SLR is “issue reports.” The term “issue reports” can be often referred to as “bug reports.” In short, the terms “issue” and “bug” should be included in our search queries.
- *Intervention:* The intervention is “requirement traceability.” The commonly used term for the “requirement traceability” studies is “trace link.” We thus used a general term “traceability” as well as “trace link.”
- *Comparison:* Since there is no alternative approach for the intervention, we did not consider this comparison part in the construction of search terms.
- *Outcome:* The outcome can be “method,” “approach,” “technique,” or “tool.” However, we did not want to limit the outcome. Considering that researchers could focus on other aspects, such as “model,” “metric,” “guideline,” “checklist,” “template,” “strategy,” etc, we did not include the outcome terms in our search queries.

We formed queries with key terms of population and intervention. The population terms are “issue” and “bug.” The intervention terms are “traceability” and “trace link.” We combined these terms with OR and AND operators in queries. Table 1 shows the queries we applied to the digital libraries. We adjusted the terms according to search results.

TABLE 1. Selected sources.

Library	Queries
IEEE Xplore	((“issue”) OR (“bug”)) AND
Google Scholar	(“trace link” OR “traceability”)
ACM Digital Library	(“issue” OR “bug”) AND
Springer Link	(“trace link” OR (“requirement
Science Direct	traceability”))
DBLP	trace\$( issue   bug)

TABLE 2. Selected studies.

Library	Total	Title & Abstract	Full-text
IEEE Xplore	124	21	9
ACM Digital Library	36	5	1
Springer Link	550	23	7
Science Direct	457	22	3
DBLP	80	29	5
Google	100	21	4
Total	1,347	121	29

2) DIGITAL LIBRARIES

We selected digital libraries according to the suggestions in [63]. We also considered renowned conferences and journals in the field of software engineering and selected the digital libraries that include such publications. As a result, we identified four digital libraries: IEEE Xplore, ACM Digital Library, Springer Link, and Science Direct. To find the key I-RT papers, we searched for papers that were published from 2017 to 2022 through the four digital libraries.

To broaden the scope of data sources, we also identified two general data sources: DBLP, and Google Scholar. Because DBLP is a data source that lists only computer science bibliography, we increased the search period and searched for the papers that were published from 2011 to 2022 through DBLP. Meanwhile, Google Scholar is a data source that promptly lists the latest publications. Therefore, we mainly searched the papers that were published from 2020 to 2022 through Google Scholar.

With the queries formed in Section III-B.1), we searched papers in the six digital libraries. As a result, we found a total of 1,347 papers, as shown in the second column of Table 2.

### 3) INCLUSION AND EXCLUSION CRITERIA

To select primary papers from the 1,347 identified papers, we defined the Inclusion and Exclusion (I/E) criteria as follows:

- Inclusion criteria:
  - Papers published from 2011 to 2022;
  - Papers that were available in full text;
  - Papers written in English.
- Exclusion criteria:
  - Papers that were classified as gray literature (e.g., posters, books, and technical reports);
  - Papers that were not related to software (e.g., food, medicine, and agricultural papers);
  - Papers that were secondary or tertiary studies (e.g., SLRs, SMSs and surveys);
  - Papers that did not mention issue (or bug) reports.

To apply the I/E criteria, we manually checked the titles and abstracts of the papers. As a result, we selected a total of 121 papers, as shown in the third column of Table 2

### 4) CHECKING FULL TEXT

We examined the full texts of the 121 papers and identified 29 papers related to our purpose, as shown in the last row of the fourth column of Table 2.

### 5) SNOWBALLING

We applied the forward and backward snowballing techniques to find additional papers [64]. For the forward snowballing method, we checked papers that cited each selected paper. To find the papers that cited each selected paper, we used Google Scholar. As a result, we identified 5 papers by using the forward snowballing method.

For the backward snowballing method, we checked papers that were cited by the selected paper. Since the snowballing method was used to identify papers strongly related to our research topic, we did not establish a limit regarding the publication date. As a result, we identified 8 papers by the backward snowballing method. After this snowballing step, we were able to identify 13 additional papers related to I-RT, as shown in Table 3.

## C. ANALYZING THE SELECTED STUDIES

We first analyzed papers with structures consisting of an Introduction, Methods, Results, Analysis, and Discussion (IMRAD). We then extracted information from a paper to answer each RQs.

### 1) ASSESSING THE QUALITY OF THE STUDIES

To assess the quality of the selected studies, we prepared our checklist by following the IMRAD structure [65].

- *Introduction*: Does the study discuss the topic of I-RT?
- *Methods*: Does the study propose methods or experiments for I-RT?
- *Results*: Does the study include novel discoveries and useful results?

- *Analysis*: Does the study analyze related studies?
- *Discussion*: Does the study discuss the pros and cons of the research area in general and identify meaningful points for future studies regarding the topic of I-RT?

After applying the IMRAD structure to the 42 studies, we found that most papers were of good quality and they were in line with our research purpose. If we estimate the score of each study by assigning 1 score when a question has a full answer with regard to the selected study, the score of the papers is above 3 out of 5. Meanwhile, 2 of the 13 studies did not propose novel methods, so we excluded these two papers [66], [67]. Finally, we identified 40 papers [16], [24], [25], [26], [27], [28], [29], [30], [31], [32], [33], [34], [35], [36], [37], [38], [39], [40], [41], [42], [43], [44], [45], [46], [47], [48], [49], [50], [51], [52], [53], [54], [55], [56], [57], [58], [59], [60], [61], [62] that were related to I-RT. Table 3 summarizes the paper selection results.

### 2) ANALYZING THE EXTRACTED INFORMATION

We extracted information from the selected papers to answer our RQs. As a result, we created the data extraction form shown in Table 4. The form included the basic information related to the RQs. We completed a form for each paper and analyzed the contents to answer each RQ.

## IV. RESULTS

### A. WHAT PROBLEMS ARE TARGETED? (RQ1)

To answer RQ1, we need a framework to analyze the problems of issue-based requirement traceability studies. Cleland-Huang et al. [15] classified traceability-related studies according to the elements of the process: planning, creating, maintaining, and using traceability. Considering the characteristics of issue-based requirement traceability studies, we identified four main problem categories as follows:

- *Trace Link Generation*: Studies focused on generating trace links between issues and other artifacts, given no previous trace links.
- *Trace Link Recovery*: The studies focused on additionally generating or repairing the trace links, given previous trace links between issues and other artifacts.
- *Trace Link Maintenance*: Studies focused on maintaining and retaining trace links as requirements change or trace links became outdated.
- *Trace Link Aid*: Studies did not directly generate, recover or maintain trace links but proposed additional techniques to assist existing traceability techniques.

Figure 2 shows the classification results of the 40 studies. What stands out from the results is that 50% of studies are related to the trace link recovery problem. We observed that there are many studies aimed at recovering trace link issues and commits. Based on our observation, the noticeable percentage makes sense. Other results are as follows. Trace link generation occupied 30%. Trace link maintenance and trace link aid took 12% and 8%, respectively.

TABLE 3. Study selection results.

Division	Venue	Searching	Snowballing	IMRAD	Total	Studies
Conference	ICSE	3	1		4	[24] [25] [26] [27]
	ASE	2	1		3	[28] [29] [30]
	RE	2	0		2	[31] [32]
	SANER	3	0		3	[33] [34] [35]
	ESEC /FSE	0	3	-1	2	[36] [37]
	REW	2	0		2	[38] [39]
	MSR	0	2		2	[40] [41]
	AIRE	0	1		1	[42]
	ICPC	0	1		1	[43]
	SBES	0	1	0	1	[44]
	ICSME	0	1		1	[45]
	WCRE	1	0		1	[46]
	ECSA	1	0		1	[47]
	REFSQ	2	0		2	[48] [49]
	EASE	1	0		1	[50]
	APSEC	1	0		1	[16]
TEFSE	0	1		1	[51]	
ICPS	1	0	-1	0		
Journal	EMSE	4	0		4	[52] [53] [54] [55]
	IST	4	0		4	[56] [57] [58] [59]
	TSE	1	0		1	[60]
	JSS	1	0		1	[61]
	KIS	0	1		1	[62]
Total	-	29	13	-2	40	

TABLE 4. Data extraction form.

Data item	Value
Paper title	What is the name of the paper?
Year	When was the paper published?
Venue	Where was the paper published?
Problems	What problems were targeted in the studies? (RQ1)
Artifacts	Which artifacts were used in the studies? (RQ2)
Techniques	What techniques were used in the studies? (RQ3)
Evaluation Targets	What evaluation targets were used for the proposed approach? (RQ4)

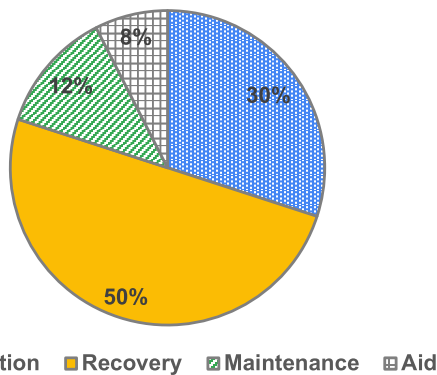


FIGURE 2. Traceability problem categories.

We then classified the investigated studies according to the four problem categories and their specific challenges, as shown in Table 5. The challenges of the studies can be grouped into five categories: accuracy, effort, support, information, and trustworthiness.

1) ACCURACY

Studies sought to improve the low accuracy of the existing techniques for generating, recovering, or maintaining trace links [24], [35], [36], [45], [48], [49], [53], [55]. Researchers attributed the low accuracy to error-prone links, semantic

gaps, and large source files. In more detail, researchers have discussed that trace links are error-prone because developers manually managed or classified them [28], [30], [33], [40], [44], [49], [58]. Researchers have also addressed that software artifacts to be linked are semantically different [24], [37], [40], [56], [61]. For instance, the study in [56] addressed that locating bugs is difficult because issue reports were written in natural language but not in source code. In addition, researchers addressed that handling large source files and stack traces was one of obstacles to improving the accuracy of trace links [45]. To improve the accuracy of trace links, researchers have also studied removing false positives from automatically generated trace links [53].

2) EFFORT

Studies have also focused on reducing a considerable amount of developers' manual effort when creating and recovering trace links [28], [33], [44], [47], [52], [53], [57], [58]. Developers spend a substantial amount of time when manually managing traceability relationships or locating bugs [29], [38], [40], [53], [58], [59], [60]. Developers also spend their time finding test cases relevant to a given bug [46]. In this regard, researchers have addressed the insufficiency of developers' inspections of the trace links between two software artifacts [27]. In particular, the study in [32] focused on improving the accuracy of effort estimation for resolving an issue, and the study in [47] addressed that recognizing and reusing architectural knowledge from issue tracking systems are challenging.

3) INFORMATION

Various studies have addressed the absence of information to create, recover, and maintain trace links. Researchers have addressed that even if developers manually link issues and commits, the trace links are often missed

TABLE 5. Problems in I-RT studies.

Problem Group	Specific Challenge	Generation	Recovery	Maintenance	Aid
Accuracy	Low accuracy	[24] [45]	[35] [36] [49] [48] [55]	[53]	
	Error-prone links	[33]	[28] [40] [44] [30] [49]	[58]	
	Semantic gap	[24]	[61] [40] [37]	[56]	
	Large source files	[45]			
	False positive links				[53]
Effort	Manual efforts	[33] [47]	[57] [28] [44] [46]	[53] [58]	[52]
	Time consumption	[60] [29] [38]	[40]	[56] [58]	[59]
	Insufficient inspection	[27]			
	Effort estimation				[32]
Information	Missing links		[26] [57] [28] [43] [44]		
	Insufficient commit messages		[36] [43] [37]		
	Lack of integration between systems		[51] [16] [50]		
	Ignorance of link types		[42] [39]		
	Lack of structural information	[29]			
	No consideration of code base	[38]			
	Ignorance of bug fixing histories	[62]			
	Ignorance of non source files in commits		[57]		
	Insufficient discriminative information		[34]		
	no overviews in dependencies of issues				[31]
	Incomplete picture of developer contributions				[54]
Support	Language issues	[25] [41]			
	Tracing other repositories and tools	[27]			
Trustworthiness	Incomplete and untrustworthy links		[26]		
	Lack of sufficient bias controls				[52]

[26], [28], [43], [44], [57]. Researchers have also addressed insufficient commit messages, which makes it difficult to identify the trace links between bug reports and commits or leads to biased defect information. [36], [37], [43]. Researchers have discussed the lack of integration between revision control systems and issue tracking systems, which affects the recovery of trace links, as well as the prediction of software faults [16], [50], [51]. Researchers addressed that issues form a complex network by themselves, but there were no approaches to predict link types [39], [42].

Regarding the specific information for generating trace links, Saha et al. addressed the lack of structural information because existing IR-based techniques handle source code as flat text [29]. Mayr-Dorn et al. addressed the problem of not considering a part of source code that actually implements a specific requirement (i.e., issue) or is covered by tests [38]. Wang et al. found that existing IR methods usually ignored the existing bug fixing histories and various sources of information (e.g., the metadata or the stack traces in the bug reports) [62].

Regarding the specific information for recovering trace links, Sun et al. noted that existing approaches disregarded nonsource files and the roles of source files in commits [57]. Nguyen-Truong et al. addressed insufficient discriminative information, which arises when existing techniques mine only the data within a commit [34].

Regarding the specific information for maintaining trace links, Luders et al. noted that it is difficult to maintain an overview of dependencies among issues [31]. Çetin and Tüzün focused on identifying the contributions of developers based on traceability graphs [54].

4) SUPPORT

Studies have attempted to develop support for languages, tools, etc., for trace links. First, researchers have addressed

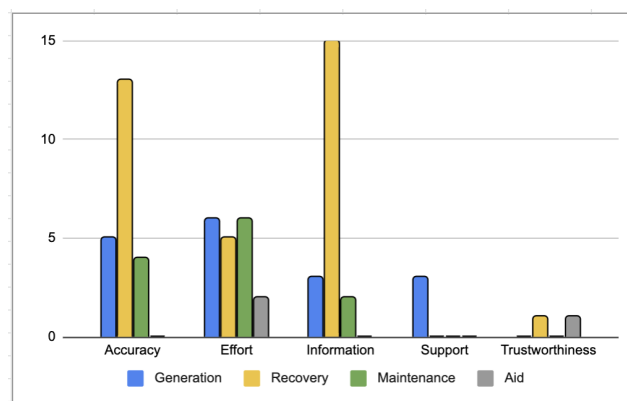


FIGURE 3. Traceability main challenges of I-RT studies.

the problems of trace links between different languages, such as natural languages [25] or intermingled languages (e.g., English, Chinese, Korean) [41]. Researchers have also addressed that existing approaches do not provide support for artifacts and traces to other repositories and tools [27].

5) TRUSTWORTHINESS

There is a challenge including incomplete and untrustworthy trace links [26]. Additionally, researchers have addressed the lack of sufficient bias control for misclassified bugs, tangled commits, and localization hints [52].

6) SUMMARY

Figure 3 shows the summary of the challenges per each problem category. We can observe that many researchers focused on the accuracy of trace links, human effort to establish trace link, and additional information to set up trace links. Especially for recovering trace links, researchers tried to use additional information to improve the accuracy of trace

links. Besides, a few researchers conducted their research on support for trace links and trustworthiness of trace links.

**RQ1:** I-RT studies can be classified into four traceability problem categories: trace link generation, recovery, maintenance, and aid. Across the four problem categories, the main challenges are low accuracy, high effort, insufficient information and support, and untrustworthiness.

## B. WHICH SOFTWARE ARTIFACT PAIRS ARE LINKED? (RQ2)

We surveyed I-RT studies and found that the studies used issue or bug reports as the primary artifacts. Our analysis for RQ2 reveals that researchers focused on linking issue reports with commits, source code, user (app) reviews, test cases, model changes, user manual, and the issue reports themselves. Table 8 shows the number of studies that handle different types of artifact pairs.

### 1) ISSUE REPORTS AND COMMITS

A total of 18 studies linked issue reports to commits [16], [24], [26], [28], [33], [34], [35], [36], [37], [41], [43], [47], [50], [53], [54], [55], [57], [61]. Among them, eleven studies focused on recovering the missing trace links between issue reports and commits [16], [26], [28], [35], [36], [37], [43], [50], [55], [57], [61]. Two studies focused on generating trace links between issue reports written in natural language and commits written in programming language [24], [41]. Five studies sought different directions. The first one used issue IDs in commit messages to assign interaction logs to requirements [53]. The second one suggested recommending an issue number so that developers could tag commit messages with the recommended issue number [33]. The third one sought to detect vulnerability-fixing commits based on the issues linked to the commit [34]. The fourth one sought to generate trace links between architectural issues and code changes [47]. The last source discussed identifying developer contributions by building traceability graphs of developers based on issues and commits [54].

### 2) ISSUE REPORTS AND SOURCE CODE

Seven studies linked issue reports to source code [29], [40], [45], [51], [52], [56], [62]. Two studies sought to aid in generating trace links between issue reports and source code [40], [52]. The studies in [45], [56], [62] sought to localize the relevant buggy source files based on the given issue (bug) reports. One study took issues and source code files as inputs to localize bugs [29], and another linked issue reports to the source code in patches [51].

### 3) ISSUE REPORTS AND TEST CASES

Four studies linked issue reports to test cases [30], [38], [44], [46]. One study focused on establishing trace links between requirements and test cases by using issue reports as requirements [38]. Three studies recovered the trace links

between issue reports and test cases [30], [44], [46]. The studies intended to link bugs and test cases.

### 4) ISSUE REPORTS

Three studies focused on the trace links among issue reports [31], [39], [42], [48]. The studies in [39], [42] focused on the different kinds of trace links, such as related links, duplicates, and blocks. The study in [31] refined the relationships of the issue reports (e.g., parent-child, duplicate, dependency, similarity, and work breakdown). The study in [48] noticed that the “related” field of an issue report contains several issue numbers, but those issues are traced because of different reasons, such as duplicate or generic ones.

### 5) ISSUE REPORTS AND USER REVIEWS

Two studies [25], [60] sought to generate trace links between user reviews and issue reports. Among them, the study in [25] tracked the states of user reviews by narrowing the gap between user reviews and issue reports. The study in [60] aimed to identify issue reports that were related to user reviews and utilized these issue reports to identify the source locations to change.

### 6) ISSUE REPORTS AND MODEL CHANGES

The study in [49] sought to recover the relationship of artifacts between Jira issues (user stories or bugs) and model changes (revisions in a Model-Driven Development (MDD) context).

### 7) ISSUE REPORTS AND USER MANUAL

The study in [58] paid attention to classifying issue reports according to specific software feature descriptions in a user manual.

### 8) ALL RELEVANT ARTIFACTS

The study in [32] traced issues to requirements, model elements, source code, texts, copies, wireframes, and art designs.

Two studies did not specify pairs of artifacts [27], [59] because the study did not focus on specific trace links. The first studies developed a tool, TimeTracer, that supports arbitrary artifacts and traces by providing APIs [27]. The second study proposed a method for automatically recommending reviewers to review various artifacts, such as requirements, design diagrams, changesets, code reviews, test cases, and bugs [59].

### 9) SUMMARY

Many researchers studied on recovering trace links between issue reports and commits and improving the accuracy of the trace links. Researchers broadened the scope of trace links to source code, test cases, and issue report themselves. Besides, a few researchers conducted studies to establish trace links between issue reports and other artifacts such as user review, model changes, and user manuals.

TABLE 6. Artifact pairs in I-RT studies.

Artifact 1	Artifact 2	Studies	Total
Issue (Bug) reports	Commits	[24] [26] [35] [61] [53] [57] [28] [41] [36] [43] [37] [33] [34] [55] [47] [16] [50] [54]	18
	Source code	[52] [56] [40] [29] [45] [51] [62]	7
	Test cases	[38] [44] [30] [46]	4
	Issue reports	[42] [31] [48] [39]	4
	User (App) reviews	[25] [60]	2
	Model changes	[49]	1
	User manual	[58]	1
	All relevant artifacts	[32]	1

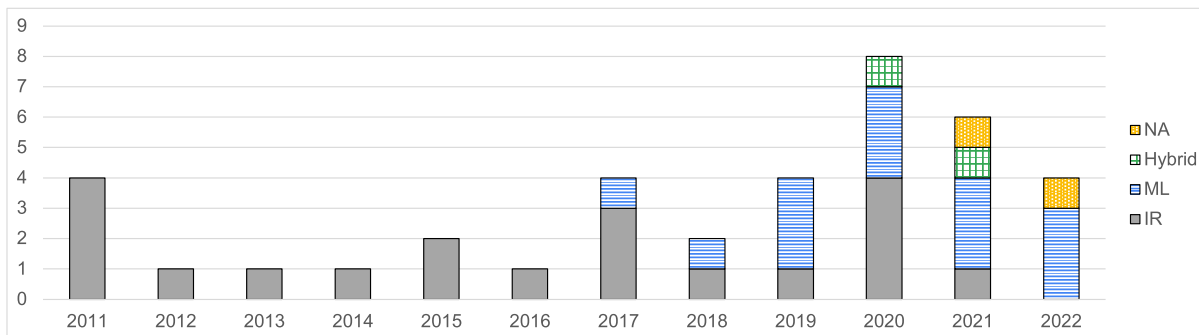


FIGURE 4. Techniques used in I-RT studies.

**RQ2:** I-RT studies linked issue reports to commits, source code, test cases, issue reports themselves, user reviews, model changes, user manual, and all other relevant artifacts.

**C. WHAT TECHNIQUES ARE USED IN THE STUDIES? (RQ3)**

We found that I-RT studies have utilized Information Retrieval (IR) and Machine Learning (ML) techniques or even hybridized them. Figure 4 shows the changes in the techniques over the years. In the early years, several studies applied IR techniques to address the challenges of generating and recovering traceability links, while recently, the use of ML techniques has gradually emerged. Table 7 summarizes the specific techniques by year.<sup>1</sup>

**1) INFORMATION RETRIEVAL-BASED APPROACHES**

From 2011 to 2016, ten related studies used IR techniques [16], [29], [36], [37], [43], [45], [46], [48], [50], [51]. The studies in [16], [36], [43], [51] sought to recover the missing links between issue reports and commits automatically. For instance, the study in [36] used the Term Frequency-Inverse Document Frequency (TF-IDF) similarity and three features. The study in [51] retrieved patches from issues, extracted patches, and recovered trace links. The study in [43] trained a random forest model with 9 text features and 11 metadata features extracted from issue reports and commit links after summarizing commit messages with ChangeScribe. The study in [50] used the SZZ algorithm proposed by Śliwerski et al. [68]. The study in [48] applies stemming, stop word removal, and term

<sup>1</sup>We did not include the study in [55] in the RQ3 results because the study in [55] did not propose a new approach, but it compared 10 different approaches.

weighting of textual data in an issue tracking system to improve the effectiveness of IR approaches for building traceability between issues. Interestingly, the study in [46] recommended test cases relevant to bugs by applying two topic modeling techniques, Latent Semantic Indexing (LSI) and Latent Dirichlet Allocation (LDA).

The studies in [29], [37], [45] sought to improve the accuracy of trace links. The study in [29] used structured IR to locate bugs by using an IR toolkit Indri automatically. The study in [37] extracted textual features by analyzing error records and change logs and passed these features to multiple detectors (e.g., pattern-based link detectors) in different layers according to their categories. The study in [45] calculated the similarities between code segments and an issue report.

In 2017, three studies used IR techniques [38], [56], [57]. Study [56] identified buggy source code files based on the similarities. Study [57] modeled the similarity distances among features extracted from issue reports, commits, source code, and documents. Study [38] identified the source code lines that implement issue reports and the test cases that cover these source code lines. Meanwhile, in 2018, one study used the IR technique [40], which utilized existing project history enriched with previously unused information to recover traceability between source code files and bug reports and increase localization performance.

In 2018 and 2019, two studies used IR techniques [31], [40]. The study in [40] intended to link issue reports and commits. To recover the traceability links between bugs and source code, the proposed approach, TestScore, selects artifacts from project histories, calculates textual similarity, constructs a graph for traceability, and finally calculates a score for each source file. The study in [31] visualized



TABLE 7. Techniques used in I-RT studies.

Years	Type	Approach	Specific Techniques	Studies
2011	IR	ReLink	Similarity	[36]
		BugTrace	Patch Retrieval, Patch Analysis, Link Recovery	[51]
		-	Latent Semantic Indexing (LSI) and Latent Dirichlet Allocation (LDA)	[46]
		-	Fellegi and Sunter (a probabilistic approach to solve record linkage problem based on decision model)	[16]
2012	IR	MLink	Pattern-Based Detection, Patch-Based Detection, Name-Based Detection, Text-Based Detection, Association-Based Detection	[37]
2013	IR	BLUiR	Abstract Syntax Tree, Eclipse Java Development Tools	[29]
2014	IR	BRTTrace	BugLocator, Revised VSM (rVSM), Source Code Segmentation, Length Function, Stack Trace Analysis, Scoring	[45]
2015	IR	RClinker	ChangeScribe, Random Forest	[43]
		-	SZZ algorithm	[50]
2016	IR	-	Stemming, stop word removal, and term weighting	[48]
2017	IR	BLIA	Scoring, rVSM	[56]
		FRLink	VSM, Natural Language Processing (NLP), Learning Algorithm, Similarity	[57]
		ReTeCo	Coverage	[38]
2017	ML	PULink	Random Forest	[28]
2018	IR	TraceScore	Scoring	[40]
	ML	Link Classifier	Naïve Bayes, J48 Decision Tree, Random Forest	[26]
2019	IR	OpenReq	Similarity	[31]
	ML	DeepLink	CBOW, NLP, Code Knowledge Graph, MLP, Support Vector Machine (SVM)	[35]
		DeepLink	NLP, Pattern, Skip-Gram, Long Short-Term Memory (LSTM)	[61]
2019	ML	Where2Change	Hierarchical Dirichlet Process (HDP), NLP, Skip-gram	[60]
2020	IR	IL <sub>COM</sub>	Tracing Interaction Events	[53]
		-	Network Analysis, Network Graphs	[42]
		TimeTracer	Data model	[27]
		STMLocator+	Latent Dirichlet Allocation (LDA), Similarity	[62]
	ML	-	NLP, Genetic Algorithm (GA)	[52]
		GVSVM	Translation, VSM, Word Embedding	[41]
IR&ML	SpojitR	Random Forest Classifier	[33]	
IR&ML	-	Latent Semantic Indexing (LSI), LDA, Best Match 25 (BM25), Word Embedding, Similarity, CNN	[44]	
2021	IR	RSTrace+	Traceability graphs	[59]
	ML	T-BERT	(Bidirectional encoder representations from transformers) BERT, NLP	[24]
		DeepMatcher	BERT, NLP	[25]
	-	-	Logistic Regression, Random Forest, Neural Network, TF-IDF	[39]
	IR&ML	-	LSI, LDA, BM25, Word Embedding, Similarity, NLP	[30]
-	-	Dependency graph	[47]	
2022	ML	LCDDTrace	Random Forests, Gradient Boosted Decision Trees	[49]
		-	Word embedding, CNN (RNN)	[58]
		HERMES	Classifiers	[34]
	-	TracIMO	Design Science	[32]
	-	-	Social Network (NetworkX package)	[54]

trace links in a graph map, detected duplicated, missed, or unknown links, and checked the inconsistencies between a release plan and the issues in a map.

In 2020, four studies used IR techniques [27], [42], [53], [62]. To create an interaction-based trace link, the study in [53] extracted the issue IDs and code entities that include developers' interactions from commits. The study in [42] investigated network analysis techniques to generate and maintain the trace links among issue reports. The study in [27] used a data model generated by using Jira and Jama data to replay trace links. The study in [62] proposed a generation model, STMLocator+, that adopts two topic models. One is based on the LDA and LLDA models, and the other captures the semantic and textual similarity.

In 2021, one study proposed using traceability graphs for recommending code reviewers [59].

## 2) MACHINE LEARNING-BASED APPROACHES

In 2017 and 2018, two studies used ML techniques [26], [28]. The study in [28] distinguished positive or unlabeled trace links, extracted features from each link, and

trained a random forest model to identify positive links. The study in [26] identified 18 features and experimented with several classification techniques, such as Naïve Bayes, J48 decision tree, and random forest classifiers.

In 2019, three studies used ML techniques [35], [60], [61]. The studies in [35], [61] proposed two approaches that are different but have the same name, DeepLink. The first DeepLink [35] used a Gated Recurrent Unit (GRU) with a Continuous Bag Of Words (CBOW) for issue report embedding. DeepLink created a graph-based code-based Abstract Syntax Tree (AST) and used a Recurrent Neural Network (RNN) to embed committed source code. The second DeepLink [61] used LSTM with skip-grams to make a fixed vector and recovered trace links by calculating the cosine similarities among (commit log, issue title), (commit log, issue description), and (commit code, issue code) pairs. To identify the code entities for an app review, the study in [60] clustered app reviews using hierarchical dirichlet processes with Natural Language Processing (NLP) and created trace links between clusters and issue reports by calculating similarity values based on skip-grams.

In 2020, three studies focused on the accuracy of bug localization based on ML techniques [33], [41], [52]. Study [52] proposed a near-optimal technique to generate a query from issue reports using a genetic algorithm. Study [41] used a Generative Vector Space Model (GVSM) to generate trace links between issue reports and commits written in two or more languages. The study [33] recommended a possible issue ID list for a new commit with a random forest classifier that is trained with features extracted from issues, commits, and source code.

In 2021, three studies used ML techniques [24], [25]. Study [24] utilized a pretrained BERT code model to retrieve code entities according to code descriptions and concatenate issue reports and commits. Study [25] used a context-sensitive text embedding method to convert app reviews and issue reports into the same vector space and used the cosine similarity metric to match app reviews with issue reports. The study [39] used several machine learning techniques with TF/IDF to predict the link types of issues.

In 2022, all three related studies utilized ML techniques [34], [49], [58]. The study in [49] proposed a machine learning classifier based on random forests and gradient boosted decision trees to classify the validity of trace links. The study in [58] proposed a deep learning model-based method based on a word embedding technique using a CNN (or RNN). The study in [34] automatically identified vulnerability-fixing commits by using 3 independent classifiers (i.e., commit message classifier, code change classifier, issue classifier).

### 3) IR-BASED & ML-BASED APPROACHES

In 2020 and 2021, two studies used a hybrid of IR and ML techniques [30], [44]. The study in [44] combined similarity metrics with LSI, LDA, BM25, and convolutional neural networks. Additionally, the study in [30], a follow-up study to [44], also used the same techniques (i.e., LSI, LDA, BM25, etc.) and vectorized issue reports as input queries and test cases as target sources. The cosine similarities between bug reports and test cases were calculated.

### 4) OTHER APPROACHES

Three studies did not use IR or ML techniques. The study in [47] constructed a dependency graph from source code, compared two consecutive versions, and determined the added or removed dependencies among classes and packages. Based on the calculation, the study attempted to link architectural issues to code changes. The study in [32] used the design science research method proposed by Wieringa [69] to propose a framework for designing traceability strategies. The study in [54] used the NetworkX package to analyze the social network of developers based on issues and commits.

### 5) SUMMARY

Figure 5 visualizes the development of techniques with time. Before 2018, researchers adopted traditional IR/ML

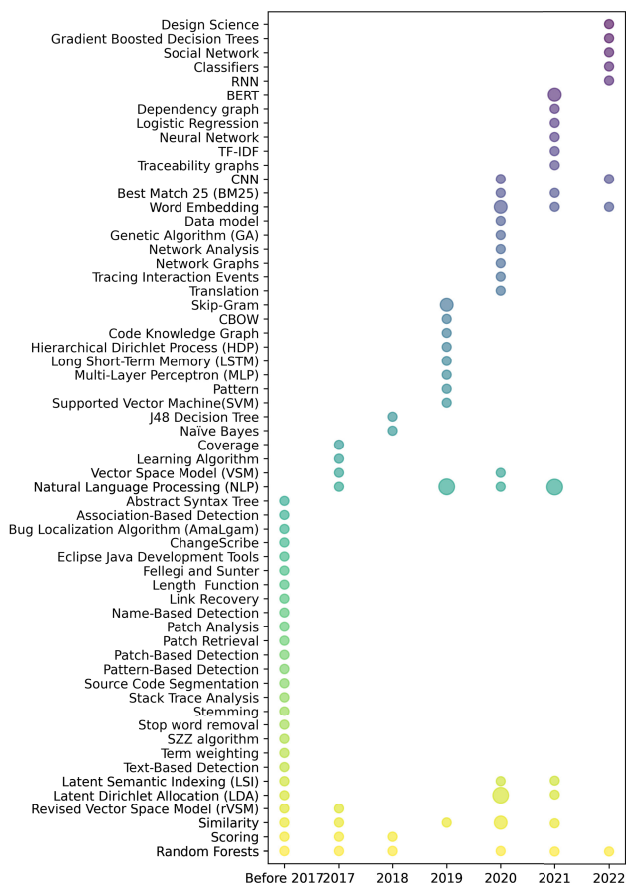


FIGURE 5. Techniques used in I-RT studies.

techniques such as Vector Space Model (VSM), similarity, random forest, and scoring. After 2018, researchers began to adopt deep learning techniques such as word embedding, Bidirectional Encoder Representations from Transformers (BERT), Convolution Neural Network (CNN), and Recurrent Neural Network (RNN) techniques.

**RQ3:** I-RT studies used ML and IR techniques. Recent studies have increasingly concentrated on ML techniques.

### D. WHAT EVALUATION TARGETS ARE USED IN THE EVALUATIONS? (RQ4)

We investigated evaluation methods and found that I-RT studies mainly conducted experiments to evaluate approaches. We then focused on evaluation targets that were used by the studies. Table 8 summarizes the evaluation targets by classifying them into three groups: open-source projects, open datasets, and student projects.

#### 1) EXPERIMENTS WITH OPEN-SOURCE PROJECTS

Among the studies that used open-source projects for their evaluation, twelve used Apache projects. [16], [28], [33], [35], [36], [37], [38], [40], [43], [47], [57], [59]. The study in [40] used 15 open-source projects to evaluate

TABLE 8. Evaluation targets used.

Target Type	Domain	Evaluation Targets	#Targets	Studies
Open Source Projects	Apache Software Foundation	Axis2, Derby, Drools, Hadoop, HornetQ, Infinispan, Iz-pack, Keycloak, Log4J2, Pig, Railo, Seam2, Teiid, Weld Wildfly	15	[40]
		Avro, Buildr, Chukwa, Falcon, Giraph, Ivy, Knox, Log4net, Nutch, OODT, Tez, Tika Zookeeper, Mahout, Chukwa, Avro, Lang, Tez	12	[28]
		Commons-CLI, Commons-Collections, Commons-CSV, Commons-IO, Commons-Lang, Commons-Math	6	[43], [57]
		Apache crunch, Apache falcon, Apache avro, Apache pig, Apache kafka	5	[33]
		Qt 3D Studio, Qt Creator, Apache Hive, Apache Zookeeper	4	[59]
		ZXing, OpenIntents, Apache	3	[36], [37]
		Apache Hadoop, Cassandra, and Tajo	3	[47]
		Ops4j paxweb, Apache qpid	2	[38]
		Apache HTTP Server, WikiMedia	2	[16]
		Eclipse, AspectJ, SWT, ZXing	4	[29]
	Eclipse	AspectJ, SWT, ZXing	3	[56]
		Eclipse, AspectJ, SWT	3	[45]
		PDE, Platform, and JDT	3	[62]
		Eclipse, Rhino	2	[51]
	Various domains	AspectJ, Birt, KookKeeper, Derby, JodaTime, Lucene, Mahout, OpenJpa, Pig, Solr, SWT, Tika, Tomcat, ZooKeeper, ZXing	15	[52]
		Keycloak, Checkstyle, Pentaho kettle, GoCD, fabric8, Closure Compiler, Flink, GRPC, Beam, CrateDB	10	[61]
		Maven, Derby, Infinispan, Groovy, Pig, Drools	6	[26]
		Avro, Tez, ZooKeeper, Chukwa, Knox	5	[55]
		Geo, Lighttpd, Rediant, and Redmine	4	[48]
	Mobile Apps	AntennaPod, Automattic, Cgeo, Chrislacy, K-9 Mail, OneBusAway, Twidere, UweTrottmann, WhisperSystems, Wordpress	10	[60]
		Firefox, VLC (media player), Signal (messenger), NextCloud (cloud storage)	4	[25]
	Mozilla	Mozilla Firefox	1	[30], [44]
	Jira	Open source projects from Jira but no descriptions of the specific projects	66	[39], [42]
	Dronology	Dronology	1	[27]
Source code editor	Notepad+, Komodo, VSCode	3	[58]	
Unknown	Samples of OSS projects from GitHub but no descriptions of the specific projects	344	[50]	
Open Datasets	SEOSS 33	datasets for 33 OSS projects (Six of 33 projects were used.)	1	[54]
	Github	CodeSearchNet, [Pgcli, Flask, and Keras]	2	[24]
	Multilingual Datasets	Konlpt, Cica, Aws-berline	3	[41]
	Mendix Studio Low-Code Platform	Service, Data, Store	3	[49]
	SAP	SAP manually-curated dataset	1	[34]
Industrial Projects	Web	A web-application developed in a company	1	[32]
	Unknown	A small industrial case study which is anonymous due to confidential reasons	1	[46]
Student Projects	Others	A Student Project that Contained 395 Commits, 40 Java Files, and 26 XML Files	1	[53]

their proposed approach (TraceScore) with state-of-the-art approaches (SimiScore and CollabScore). The study in [28] used 12 projects to evaluate their proposed FULink approach with FRlink. The studies in [35], [43], [57] collected true links (i.e., commits that fixed issue reports) from open-source projects. The study in [33] listed 5 open-source projects but only used the Apache crunch project to evaluate the recommendations of issue IDs for comments. The study in [59] used 4 projects to compare their method with 3 other methods, Naive-Bayes, RevFinder, and Profile. Three studies used 3 projects [36], [37], [47]. Two studies in [36], [37] evaluated the accuracy of the trace links between issue reports and commits or bug locations. One study in [47] estimated the size of architectural changes based on architectural issues and code changes. Two studies used two projects [16], [38]. The study in [38] demonstrated the capability of the proposed approach (ReTeCe) to provide requirement coverage reports. The study in [16] collected links between commits and issues and used the links to evaluate the proposed approach in terms of precision.

Next, five studies used projects belonging to Eclipse. [29], [45], [51], [56], [62]. Studies used 34 projects to

evaluate the accuracy of the trace links between issue reports and commits [29], [45], [56]. The study in [62] selected 3 projects from the official Bug Tracking Website of Eclipse. The study in [51] used 2 projects to evaluate the proposed approach, BugTrace, where the study compared automatically recovered trace links with manually recovered links.

Five studies selected various open-source projects. [26], [48], [52], [55], [61]. The study in [52] used 803 issue reports from 15 open-source projects. The study in [61] selected 10 projects from 1,078 Java projects. The study in [26] selected 6 projects, where the study collected trace data from project management systems, issue tracking systems, and code management systems. The study in [55] selected 5 projects to comparatively evaluate 10 issue-linking algorithms. The study in [48] selected 4 projects and extracted 100 consecutive issues per project.

Two studies used mobile app projects. The study in [60] used 10 projects where experts built the ground truth as an answer set. The study in [25] used 4 projects, where the study randomly sampled 50 app reviews for each app, manually verified them, and linked them to issue reports for evaluation.



### C. GENERATING OR RECOVERING TRUSTWORTHY TRACE LINKS

Many IR and ML techniques have been used to create and recover the trace links between issue reports and other software artifacts. However, the trace links automatically recovered by existing techniques are still inaccurate and untrustworthy. Thus, we question whether it would be sufficient to enhance IR and ML techniques to improve the accuracy of trace links. We need to find a way to improve the accuracy and trustworthiness of trace links or a way to inform a degree of trustworthiness of trace links. When trace links are trustworthy or a degree of trustworthiness is explicit, stakeholders will consider using the IR/ML based automated methods to generate or recover trace links.

### D. GROWTH OF OPEN DATASETS

For evaluation purposes, many I-RT studies have targeted open-source projects, utilizing their revision histories. Only a few studies used open datasets created by others, which may impede the fairness of evaluation. Thus, we encourage the creation and sharing of open datasets when researching the topic of I-RT studies for a fair evaluation and innovation of traceability research. Open datasets will help researchers experiment with different techniques to improve the accuracy of the generated trace links. In addition, the continuously enriching open datasets could increase the reliability of I-RT studies.

## VI. THREATS TO VALIDITY

Our SLR paper can face potential threats to validity, which can be divided into construct, internal, and external validity. The threats and mitigation strategies are described as follows:

### A. CONSTRUCT VALIDITY

This validity concerns the process of identifying papers. The selection results of papers depend on the coherence of our search queries. To mitigate this threat, we carefully identified search terms and adjusted the combination of the terms according to the digital libraries. We also clarified the queries we made per digital library. Additionally, we defined exclusion and inclusion criteria to review these relevant studies for exact identification. Meanwhile, we also utilized snowballing techniques to obtain as many related studies as possible.

### B. INTERNAL VALIDITY

We treated studies that used issue reports or bug reports as I-RT studies. That is, the studies could be diverse, and the studies whose main concerns were not traceability might have been included. For instance, bug localization papers could be included if the papers handled traceability of issue reports. From our point of view, such an inclusion is not a big deal because bug localization is one of the goals that traceability studies typically aim to achieve. Additionally, to mitigate this threat, we carefully determined whether the studies

were related to traceability by checking which “trace” terms appeared in the studies. Another threat to internal validity is that the four authors individually extracted and analyzed the data from the selected papers. Different participants may have different views about data, so individual analysis could affect the detailed analysis results of the paper. To mitigate this threat, we conducted weekly discussions.

### C. EXTERNAL VALIDITY

We searched papers with keywords in digital libraries by focusing on top-tier conferences and journals as a starting point. This perspective may be narrow, and their rankings may be inaccurate or slightly changed. To mitigate external threats, we conducted multiple rounds of comparative analysis when selecting them.

## VII. CONCLUSION

We conducted an SLR to investigate the trends of I-RT studies in terms of four aspects: problems, artifact pairs, techniques, and evaluation targets. We summarize our findings as follows. First, the I-RT studies addressed the challenges of low accuracy, manual effort, insufficient support and information, and untrustworthiness of trace links. Second, the artifacts linked to issue reports are commits, source code, user reviews, test cases, etc. Third, most of the techniques used in the studies were ML and IR approaches. Finally, the primary evaluation targets used in the studies were open-source projects.

With the results, we also discussed the challenges related to I-RT. Based on our discussion, we propose future research directions. First, we need additional information to improve the accuracy and trustworthiness of trace links. In our future direction, we plan to develop state-of-the-art techniques or tools to overcome the challenge of insufficient information. Second, we need to find a way to fairly evaluate I-RT approaches. Therefore, it will also be essential to build large open datasets to improve the reliability of the evaluations.

## REFERENCES

- [1] O. C. Z. Gotel and C. W. Finkelstein, “An analysis of the requirements traceability problem,” in *Proc. IEEE Int. Conf. Requirements Eng.*, 1994, pp. 94–101.
- [2] G. Spanoudakis, A. Zisman, E. Pérez-Miñana, and P. Krause, “Rule-based generation of requirements traceability relations,” *J. Syst. Softw.*, vol. 72, no. 2, pp. 105–127, Jul. 2004.
- [3] N. Ali, Y.-G. Guéhéneuc, and G. Antoniol, “Trustrace: Mining software repositories to improve the accuracy of requirement traceability links,” *IEEE Trans. Softw. Eng.*, vol. 39, no. 5, pp. 725–741, May 2013.
- [4] F. Wang, Z.-B. Yang, Z.-Q. Huang, C.-W. Liu, Y. Zhou, J.-P. Bodeveix, and M. Filali, “An approach to generate the traceability between restricted natural language requirements and AADL models,” *IEEE Trans. Rel.*, vol. 69, no. 1, pp. 154–173, Mar. 2019.
- [5] G. Antoniol, G. Canfora, G. Casazza, A. D. Lucia, and E. Merlo, “Recovering traceability links between code and documentation,” *IEEE Trans. Softw. Eng.*, vol. 28, no. 10, pp. 970–983, Oct. 2002.
- [6] D. V. Rodriguez and D. L. Carver, “Comparison of information retrieval techniques for traceability link recovery,” in *Proc. IEEE 2nd Int. Conf. Inf. Comput. Technol. (ICICT)*, Mar. 2019, pp. 186–193.
- [7] T. Hey, “INDIRECT: Intent-driven requirements-to-code traceability,” in *Proc. IEEE/ACM 41st Int. Conf. Softw. Eng., Companion (ICSE-Companion)*, May 2019, pp. 190–191.

- [8] M. Borg, P. Runeson, and A. Ardö, "Recovering from a decade: A systematic mapping of information retrieval approaches to software traceability," *Empirical Softw. Eng.*, vol. 19, no. 6, pp. 1565–1616, Dec. 2014.
- [9] T. Vale, E. S. De Almeida, V. Alves, U. Kulesza, N. Niu, and R. De Lima, "Software product lines traceability: A systematic mapping study," *Inf. Softw. Technol.*, vol. 84, pp. 1–18, Apr. 2017.
- [10] S. Charalampidou, A. Ampatzoglou, E. Karountzos, and P. Avgeriou, "Empirical studies on software traceability: A mapping study," *J. Softw., Evol. Process.*, vol. 33, no. 2, p. e2294, Feb. 2021.
- [11] N. Mustafa and Y. Labiche, "The need for traceability in heterogeneous systems: A systematic literature review," in *Proc. IEEE 41st Annu. Comput. Softw. Appl. Conf. (COMPSAC)*, Jul. 2017, pp. 305–310.
- [12] H. Tufail, M. F. Masood, B. Zeb, F. Azam, and M. W. Anwar, "A systematic review of requirement traceability techniques and tools," in *Proc. 2nd Int. Conf. Syst. Rel. Saf. (ICSRS)*, Dec. 2017, pp. 450–454.
- [13] B. Wang, R. Peng, Y. Li, H. Lai, and Z. Wang, "Requirements traceability technologies and technology transfer decision support: A systematic review," *J. Syst. Softw.*, vol. 146, pp. 59–79, Dec. 2018.
- [14] T. W. W. Aung, H. Huo, and Y. Sui, "A literature review of automatic traceability links recovery for software change impact analysis," in *Proc. 28th Int. Conf. Program Comprehension*, Jul. 2020, pp. 14–24.
- [15] J. Cleland-Huang, O. C. Gotel, J. H. Hayes, P. Mäder, and A. Zisman, "Software traceability: Trends and future directions," in *Proc. Future Softw. Eng.*, 2014, pp. 55–69.
- [16] A. Sureka, S. Lal, and L. Agarwal, "Applying Fellegi–Sunter (FS) model for traceability link recovery between bug databases and version archives," in *Proc. 18th Asia-Pacific Softw. Eng. Conf.*, Dec. 2011, pp. 146–153.
- [17] M. White, M. Linares-Vasquez, P. Johnson, C. Bernal-Cardenas, and D. Poshyvanyk, "Generating reproducible and replayable bug reports from Android application crashes," in *Proc. IEEE 23rd Int. Conf. Program Comprehension*, May 2015, pp. 48–59.
- [18] G. Schermann, M. Brandtner, S. Panichella, P. Leitner, and H. Gall, "Discovering loners and phantoms in commit and issue data," in *Proc. IEEE 23rd Int. Conf. Program Comprehension*, May 2015, pp. 4–14.
- [19] M. Salama, R. Bahsoon, and N. Bencomo, "Managing trade-offs in self-adaptive software architectures: A systematic mapping study," in *Managing Trade-offs in Adaptable Software Architectures*. Elsevier, 2017, pp. 249–297, doi: 10.1016/B978-0-12-802855-1.00011-3.
- [20] H. Snyder, "Literature review as a research methodology: An overview and guidelines," *J. Bus. Res.*, vol. 104, pp. 333–339, Nov. 2019.
- [21] B. A. Farshchian and Y. Dahl, "The role of ICT in addressing the challenges of age-related falls: A research agenda based on a systematic mapping of the literature," *Pers. Ubiquitous Comput.*, vol. 19, nos. 3–4, pp. 649–666, Jul. 2015.
- [22] F. Tian, T. Wang, P. Liang, C. Wang, A. A. Khan, and M. A. Babar, "The impact of traceability on software maintenance and evolution: A mapping study," *J. Softw., Evol. Process.*, vol. 33, no. 10, p. e2374, Oct. 2021.
- [23] B. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering," Keele Univ., Keele, U.K., Tech. Rep., 2007.
- [24] J. Lin, Y. Liu, Q. Zeng, M. Jiang, and J. Cleland-Huang, "Traceability transformed: Generating more accurate links with pre-trained BERT models," in *Proc. IEEE/ACM 43rd Int. Conf. Softw. Eng. (ICSE)*, May 2021, pp. 324–335.
- [25] M. Haering, C. Stanik, and W. Maalej, "Automatically matching bug reports with related app reviews," in *Proc. IEEE/ACM 43rd Int. Conf. Softw. Eng. (ICSE)*, May 2021, pp. 970–981.
- [26] M. Rath, J. Rendall, J. L. C. Guo, J. Cleland-Huang, and P. Mäder, "Traceability in the wild: Automatically augmenting incomplete trace links," in *Proc. 40th Int. Conf. Softw. Eng.*, May 2018, pp. 834–845.
- [27] C. Mayr-Dorn, M. Vierhauser, F. Keplinger, S. Bichler, and A. Egyed, "TimeTracer: A tool for back in time traceability replaying," in *Proc. ACM/IEEE 42nd Int. Conf. Softw. Eng., Companion*, Jun. 2020, pp. 33–36.
- [28] Y. Sun, C. Chen, Q. Wang, and B. Boehm, "Improving missing issue-commit link recovery using positive and unlabeled data," in *Proc. 32nd IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, Oct. 2017, pp. 147–152.
- [29] R. K. Saha, M. Lease, S. Khurshid, and D. E. Perry, "Improving bug localization using structured information retrieval," in *Proc. 28th IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, Nov. 2013, pp. 345–355.
- [30] G. Gadelha, F. Ramalho, and T. Massoni, "Traceability recovery between bug reports and test cases—A Mozilla Firefox case study," *Automated Softw. Eng.*, vol. 28, no. 2, pp. 1–46, Nov. 2021.
- [31] C. M. Lüders, M. Raatikainen, J. Motger, and W. Maalej, "OpenReq issue link map: A tool to visualize issue links in Jira," in *Proc. IEEE 27th Int. Requirements Eng. Conf. (RE)*, Sep. 2019, pp. 492–493.
- [32] S. Maro, J.-P. Steghöfer, P. Bozzelli, and H. Muccini, "TracIMO: A traceability introduction methodology and its evaluation in an agile development team," *Requirements Eng.*, vol. 27, no. 1, pp. 53–81, Mar. 2022.
- [33] M. Rath, M. T. Tomova, and P. Mäder, "Spojitr: Intelligently link development artifacts," in *Proc. IEEE 27th Int. Conf. Softw. Anal., Evol. Reeng. (SANER)*, Feb. 2020, pp. 652–656.
- [34] G. Nguyen-Truong, H. J. Kang, D. Lo, A. Sharma, A. E. Santosa, A. Sharma, and M. Y. Ang, "HERMES: Using commit-issue linking to detect vulnerability-fixing commits," in *Proc. IEEE Int. Conf. Softw. Anal., Evol. Reengineering (SANER)*, Mar. 2022, pp. 51–62.
- [35] R. Xie, L. Chen, W. Ye, Z. Li, T. Hu, D. Du, and S. Zhang, "DeepLink: A code knowledge graph based learning approach for issue-commit link recovery," in *Proc. IEEE 26th Int. Conf. Softw. Anal., Evol. Reengineering (SANER)*, Feb. 2019, pp. 434–444.
- [36] R. Wu, H. Zhang, S. Kim, and S.-C. Cheung, "ReLink: Recovering links between bugs and changes," in *Proc. 19th ACM SIGSOFT Symp. 13th Eur. Conf. Found. Softw. Eng.*, Sep. 2011, pp. 15–25.
- [37] A. T. Nguyen, T. T. Nguyen, H. A. Nguyen, and T. N. Nguyen, "Multi-layered approach for recovering links between bug reports and fixes," in *Proc. ACM SIGSOFT 20th Int. Symp. Found. Softw. Eng.*, Nov. 2012, pp. 1–11.
- [38] R. Mordinyi and S. Biffl, "Exploring traceability links via issues for detailed requirements coverage reports," in *Proc. IEEE 25th Int. Requirements Eng. Conf. Workshops (REW)*, Sep. 2017, pp. 359–366.
- [39] A. Nicholson and G. Jin L. C., "Issue link label recovery and prediction for open source software," in *Proc. IEEE 29th Int. Requirements Eng. Conf. Workshops (REW)*, Sep. 2021, pp. 126–135.
- [40] M. Rath, D. Lo, and P. Mäder, "Analyzing requirements and traceability information to improve bug localization," in *Proc. 15th Int. Conf. Mining Softw. Repositories*, May 2018, pp. 442–453.
- [41] Y. Liu, J. Lin, and J. Cleland-Huang, "Traceability support for multi-lingual software projects," in *Proc. 17th Int. Conf. Mining Softw. Repositories*, Jun. 2020, pp. 443–454.
- [42] A. Nicholson, D. M. Arya, and J. L. C. Guo, "Traceability network analysis: A case study of links in issue tracking systems," in *Proc. IEEE 7th Int. Workshop Artif. Intell. Requirements Eng. (AIRE)*, Sep. 2020, pp. 39–47.
- [43] T.-D.-B. Le, M. Linares-Vásquez, D. Lo, and D. Poshyvanyk, "RCLinker: Automated linking of issue reports and commits leveraging rich contextual information," in *Proc. IEEE 23rd Int. Conf. Program Comprehension*, May 2015, pp. 36–47.
- [44] L. R. J. Santos, G. Gadelha, F. Ramalho, and T. Massoni, "Improving traceability recovery between bug reports and manual test cases," in *Proc. 34th Brazilian Symp. Softw. Eng.*, Oct. 2020, pp. 293–302.
- [45] C.-P. Wong, Y. Xiong, H. Zhang, D. Hao, L. Zhang, and H. Mei, "Boosting bug-report-oriented fault localization with segmentation and stack-trace analysis," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol.*, Sep. 2014, pp. 181–190.
- [46] N. Kaushik, L. Tahvildari, and M. Moore, "Reconstructing traceability between bugs and test cases: An experimental study," in *Proc. 18th Work. Conf. Reverse Eng.*, Oct. 2011, pp. 411–414.
- [47] M. Soliman, M. Galster, and P. Avgeriou, "An exploratory study on architectural knowledge in issue tracking systems," in *Proc. Eur. Conf. Softw. Archit.* Cham, Switzerland: Springer, 2021, pp. 117–133.
- [48] T. Merten, D. Krämer, B. Mager, P. Schell, S. Bürsner, and B. Paech, "Do information retrieval algorithms for automated traceability perform effectively on issue tracking system data?" in *Proc. Int. Work. Conf. Requirements Eng., Found. Softw. Quality*. Cham, Switzerland: Springer, 2016, pp. 45–62.
- [49] R. Rasiman, F. Dalpiaz, and S. España, "How effective is automated trace link recovery in model-driven development?" in *Proc. Int. Work. Conf. Requirements Eng., Found. Softw. Quality*. Cham, Switzerland: Springer, 2022, pp. 35–51.
- [50] B. A. Romo and A. Capiluppi, "Towards an automation of the traceability of bugs from development logs: A study based on open source software," in *Proc. 19th Int. Conf. Eval. Assessment Softw. Eng.*, Apr. 2015, pp. 1–6.

- [51] C. S. Corley, N. A. Kraft, L. H. Etzkorn, and S. K. Lukins, "Recovering traceability links between source code and fixed bugs via patch analysis," in *Proc. 6th Int. Workshop Traceability Emerg. Forms Softw. Eng.*, May 2011, pp. 31–37.
- [52] C. Mills, E. Parra, J. Pantiuchina, G. Bavota, and S. Haiduc, "On the relationship between bug reports and queries for text retrieval-based bug localization," *Empirical Softw. Eng.*, vol. 25, no. 5, pp. 3086–3127, Sep. 2020.
- [53] P. Hübner and B. Paech, "Interaction-based creation and maintenance of continuously usable trace links between requirements and source code," *Empirical Softw. Eng.*, vol. 25, no. 5, pp. 4350–4377, Sep. 2020.
- [54] H. A. Çetin and E. Tüzün, "Analyzing developer contributions using artifact traceability graphs," *Empirical Softw. Eng.*, vol. 27, no. 3, pp. 1–49, May 2022.
- [55] M. Kondo, Y. Kashiwa, Y. Kamei, and O. Mizuno, "An empirical study of issue-link algorithms: Which issue-link algorithms should we use?" *Empirical Softw. Eng.*, vol. 27, no. 6, pp. 1–50, Nov. 2022.
- [56] K. C. Youm, J. Ahn, and E. Lee, "Improved bug localization based on code change histories and bug reports," *Inf. Softw. Technol.*, vol. 82, pp. 177–192, Feb. 2017.
- [57] Y. Sun, Q. Wang, and Y. Yang, "FRLink: Improving the recovery of missing issue-commit links by revisiting file relevance," *Inf. Softw. Technol.*, vol. 84, pp. 33–47, Apr. 2017.
- [58] H. Cho, S. Lee, and S. Kang, "Classifying issue reports according to feature descriptions in a user manual based on a deep learning model," *Inf. Softw. Technol.*, vol. 142, Feb. 2022, Art. no. 106743.
- [59] E. Sülün, E. Tüzün, and U. Doğrusöz, "RSTrace+: Reviewer suggestion using software artifact traceability graphs," *Inf. Softw. Technol.*, vol. 130, Feb. 2021, Art. no. 106455.
- [60] T. Zhang, J. Chen, X. Zhan, X. Luo, D. Lo, and H. Jiang, "Where2Change: Change request localization for app reviews," *IEEE Trans. Softw. Eng.*, vol. 47, no. 11, pp. 2590–2616, Nov. 2021.
- [61] H. Ruan, B. Chen, X. Peng, and W. Zhao, "DeepLink: Recovering issue-commit links based on deep learning," *J. Syst. Softw.*, vol. 158, Dec. 2019, Art. no. 110406.
- [62] Y. Wang, Y. Yao, H. Tong, X. Huo, M. Li, F. Xu, and J. Lu, "Enhancing supervised bug localization with metadata and stack-trace," *Knowl. Inf. Syst.*, vol. 62, no. 6, pp. 2461–2484, Jun. 2020.
- [63] L. Chen, M. A. Babar, and H. N. Zhang, "Towards an evidence-based understanding of electronic data sources," in *Proc. Electron. Workshops Comput.*, Apr. 2010, pp. 1–4.
- [64] C. Wohlin, "Guidelines for snowballing in systematic literature studies and a replication in software engineering," in *Proc. 18th Int. Conf. Eval. Assessment Softw. Eng.*, May 2014, pp. 1–10.
- [65] M. Saleem and N. M. Minhas, "Information retrieval based requirement traceability recovery approaches—a systematic literature review," *Univ. Sindh J. Inf. Commun. Technol.*, vol. 2, no. 4, pp. 180–188, 2018.
- [66] Y. Sun, Q. Wang, and M. Li, "Understanding the contribution of non-source documents in improving missing link recovery: An empirical study," in *Proc. 10th ACM/IEEE Int. Symp. Empirical Softw. Eng. Meas.*, Sep. 2016, pp. 1–10.
- [67] M. Gupta and A. Sureka, "Nirikshan: Mining bug report history for discovering process maps, inefficiencies and inconsistencies," in *Proc. 7th India Softw. Eng. Conf.*, Feb. 2014, pp. 1–10.
- [68] J. Śliwerski, T. Zimmermann, and A. Zeller, "When do changes induce fixes?" in *ACM SIGSOFT Softw. Eng. Notes*, vol. 30, no. 4, pp. 1–5, 2005.
- [69] R. Wieringa, "Design science methodology: Principles and practice," in *Proc. 32nd ACM/IEEE Int. Conf. Softw. Eng.*, vol. 2, May 2010, pp. 493–494.
- [70] M. Rath and P. Mäder, "The SEOSS 33 dataset—Requirements, bug reports, code history, and trace links for entire projects," *Data Brief*, vol. 25, Aug. 2019, Art. no. 104005.



**YIJING LYU** received the B.S. degree from the Department of Computer Science and Technology, Zhengzhou Normal University, China, in 2019. She is currently pursuing the master's degree with the Department of Aerospace and Software Engineering and the Department of AI Convergence Engineering, Gyeongsang National University.



**HEETAE CHO** received the B.S. degree from the Department of Aerospace and Software Engineering, Gyeongsang National University, in 2018, and the M.S. degree from the Department of Informatics, Gyeongsang National University, in 2020. He is currently pursuing the Ph.D. degree with the Department of AI Convergence Engineering, Gyeongsang National University. His research interests include software visualization, explainable AI, and verification and validation.



**PILSU JUNG** received the B.S. degree in computer science and engineering from Chungnam National University, in 2012, and the M.S. and Ph.D. degrees from the School of Computing, KAIST, in 2014 and 2020, respectively. He worked as a Staff Software Engineer with Samsung Electronics, from 2020 to 2022. Currently, he is an Assistant Professor with the Department of Aerospace and Software Engineering, Gyeongsang National University. His research interests include software

reuse, software quality, software product line engineering, software architecture, and software testing.



**SEONAH LEE** (Member, IEEE) received the B.S. and M.S. degrees in computer science and engineering from Ewha Womans University, in 1997 and 1999, respectively, the M.S.E. degree from the School of Computer Science, Carnegie Mellon University, in 2005, and the Ph.D. degree from the School of Computer Science, KAIST, in 2013. She worked as a Software Engineer with Samsung Electronics, from 1999 to 2006. She worked as a Research Professor at KAIST,

from 2014 to 2015. Currently, she is an Associate Professor with the Department of Aerospace and Software Engineering and the Department of AI Convergence Engineering, Gyeongsang National University. Her research interests include software evolution, documentation updates, requirement traceability, software architecture, and data mining.

...