Applications of natural language processing in software traceability

Pauzi, Zaki; Capiluppi, Andrea

Published in:
Journal of Systems and Software

DOI:
10.1016/j.jss.2023.111616

The Journal of Systems & Software 198 (2023) 111616

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

Applications of natural language processing in software traceability: A systematic mapping study☆

Zaki Pauzi *, Andrea Capiluppi

Bernoulli Institute, University of Groningen, Nijenborgh 9, Groningen, 9747 AG, The Netherlands

ARTICLE INFO

Article history:
Received 20 July 2022
Received in revised form 6 December 2022
Accepted 11 January 2023
Available online 16 January 2023

Keywords:
Software traceability
Information retrieval
Natural language processing

ABSTRACT

A key part of software evolution and maintenance is the continuous integration from collaborative efforts, often resulting in complex traceability challenges between software artifacts: features and modules remain scattered in the source code, and traceability links become harder to recover. In this paper, we perform a systematic mapping study dealing with recent research recovering these links through information retrieval, with a particular focus on natural language processing (NLP).

Our search strategy gathered a total of 96 papers in focus of our study, covering a period from 2013 to 2021. We conducted trend analysis on NLP techniques and tools involved, and traceability efforts (applying NLP) across the software development life cycle (SDLC). Based on our study, we have identified the following key issues, barriers, and setbacks: syntax convention, configuration, translation, explainability, properties representation, tacit knowledge dependency, scalability, and data availability.

Based on these, we consolidated the following open challenges: representation similarity across artifacts, the effectiveness of NLP for traceability, and achieving scalable, adaptive, and explainable models. To address these challenges, we recommend a holistic framework for NLP solutions to achieve effective traceability and efforts in achieving interoperability and explainability in NLP models for traceability.

1. Introduction

Software traceability is a fundamentally important task in software engineering: for some domains, traceability is even assessed by certifying bodies (Guo et al., 2017a). Given that traceability permeates all aspects of software production, the need for automated traceability has increased too, considering that software projects have steadily become more complex and the ever-increasing number of artifacts (Cleland-Huang et al., 2007; Duan et al., 2009; Guo et al., 2017b).

The underlying complexities of the logical relations between artifacts, at various stages in the software process, have prompted a variety of empirical studies (Maletic et al., 2003; Schwarz et al., 2010; Mäder et al., 2017) and several areas of research, particularly in the inception of semantic domain knowledge (Marcus and Maletic, 2003; Zhao et al., 2017a). During the software process life cycle, complex traceability challenges emerge due to differential evolution and the heterogeneity of artifacts, rendering trace link retrieval to be onerous. This calls for a *holistic* framework that requires tools and techniques to be able to promote extensibility and automation (by having a common representation), mapping of native representation to common representation, and rules defining consistency between artifacts (Pete and Balasubramaniam, 2015).

As we endeavour to achieve this framework, it is inevitable to acknowledge the role of natural language processing (NLP) in these efforts; a viable research frontier solution to traceability problems (Arunthavanathan et al., 2016). With recent advancements in NLP, we are addressing a *critical* need to consolidate and study all recent research efforts in this space.

Extracting information from a corpus of text to derive meaningful output is a technique most often found in NLP. In other words, semantic extraction is obtained from textual data and arranged in formal grammars that specify relationships between text units (Nadkarni et al., 2011). The role of NLP in software traceability addresses limitations of conventional Information Retrieval (IR), particularly around natural language data composition (Russell-Rose and Stevenson, 2009). NLP plays a vital role in these efforts, yet there is very little done to study the existing research efforts in this space. We have devised the following general topics of research focus:

1. Extracting *meaningful information* from software artifacts using NLP tools;
2. Recovering *traceability links* through automatic or semi-automatic approaches;

☆ Editor: Nicole Novielli.

* Corresponding author.
E-mail addresses: a.z.bin.mohamad.pauzi@rug.nl (Z. Pauzi), a.capiluppi@rug.nl (A. Capiluppi).

https://doi.org/10.1016/j.jss.2023.111616

3. Binding the extracted information with domain-specific concepts to decipher *context* or *domain*.

These topics form the basis and rationale for this systematic mapping study (SMS), addressing the problem of traceability recovery through solutions of information retrieval with NLP. Given the width and the breadth of traceability in the software lifecycle, an SMS is a more appropriate approach to uncover the ways in which NLP has been instrumented and deployed, and in which phase of the software life cycle. By conducting this study, we are able to consolidate diverse and scattered efforts across multiple branches, and identify key areas of gaps pertaining to traceability solutions that necessitate more attention.

The following research questions were outlined based on existing research and work in NLP for software traceability, and will be assessed as part of the SMS:

RQ1: What are the demographics of the published articles?

Rationale: This information gives us an overview of the publications' metadata, enabling impact and quality analysis. We will also analyse high-impact publications as part of our study.

RQ2: What is the trend analysis of NLP techniques and tools proposed and evaluated in the published articles?

Rationale: This allows us to establish the state of existing knowledge and efforts, subsequently allowing us to identify research gaps in our current understanding, and predict how future trends may be.

RQ3: What is the trend analysis across the phases of the SDLC?

Rationale: By using the SDLC framework, we can identify key areas of NLP application in traceability that were proposed and evaluated in publications. Given the width and breadth of the SDLC, an SMS appears to be a better choice than a Systematic Literature Review (SLR).

RQ4: What are the reported key issues, barriers and setbacks?

Rationale: Through collating these, we are able to consolidate pain points and bottlenecks. This allows us to understand the perils and pitfalls of NLP in traceability so we can identify focus areas for future research.

RQ5: What are the open challenges?

Rationale: From the key issues, barriers and setbacks identified, we collate the themes covering these as open challenges.

This paper aims to tackle these questions by conducting a thoroughly focused, yet comprehensive, systematic mapping study. This paper addresses the need to consolidate recent NLP efforts in traceability, analyse what are the common issues, barriers, and setbacks to effective traceability, and provide recommendations to address open challenges. Section 3 will explain the methodology and data process behind the study. Section 4 will cover the results and subsequently will be discussed and analysed in Section 5. Section 7 finally concludes.

2. Background

2.1. Contextual definition

NLP is a branch of Artificial Intelligence and Linguistics that allows the representation and analysis of human language computationally (Khurana et al., 2017). Due to the recent phenomenon of vast amounts of unstructured textual data being collected and used for machine learning, applications of NLP to solve real-world problems is gaining more attention from researchers and practitioners alike. In the context of software engineering, NLP is utilised to harness value from the natural language present in software artifacts. Justification of the use of the textual format of these artifacts relates to the following (Yalla and Sharma, 2015):

- possibility for automation
- information that is naturally represented, thus making it recognisable and readable for humans
- easy and practical to develop and use

By leveraging the syntactic and semantic nature of software artifacts, we aim to study past and current efforts in trace-link recovery between software artifacts that used NLP techniques and tools. Our paper looks into multiple perspectives (orientation) of software traceability, and the application of NLP to achieve the goals of traceability between software artifacts, including the 'golden challenge' of ubiquitous traceability (Cleland-Huang et al., 2014), that is, instrumenting traceability to be built into the engineering process.

2.2. Related work

A mapping study of IR approaches to software traceability was completed in 2014, with a particular focus on previous evaluations and evidence strength (Borg et al., 2014). The study; however, was done excluding core techniques in NLP methods such as machine learning (Spanoudakis et al., 2003) and semantic networks (Lindvall et al., 2009). These were disregarded in the study as they were too different to fit in the scope due to the complexities in development and deployment. However, the landscape in NLP research has witnessed major breakthroughs in recent years, driving a new wave of tools and applications specifically for software engineering tasks (Sawant and Devanbu, 2021). Some examples of applications: training word embeddings in the software engineering domain space (Efstathiou et al., 2018), requirements classification using deep learning (Navarro-Almanza et al., 2017), and textual classification of natural language in software engineering text mining pipelines (Mäntylä et al., 2018).

A more recent review was done, broadly focusing on adopting NLP to mine unstructured data in software repositories (Gupta and Gupta, 2019). The review was done by looking into general applications of mining repositories, with a sub-focus on traceability efforts. In terms of integrating NLP applications into the SDLC, an assessment of how NLP is employed (in the different phases) was mentioned in Yalla and Sharma (2015). This integration, deemed as multidisciplinary research, highlights the potential advantages: a more holistic approach to Computer Science and Engineering, greater possibility for automation, and a step closer to achieving *universal programmability*: the possibility to program in a natural language, and without the need of a formal programming language (Tichy et al., 2013). In the context of traceability in artifacts, the need for precise semantics for trace links between heterogeneous systems is critical due to inadequate available tools (Mustafa and Labiche, 2017). This review highlighted the need to define a taxonomy for trace links, as characteristics of trace data are likely to be domain-, organisation-, or even project-dependent.

Although machine learning (such as NLP) has gained an incredible amount of attention only in recent years, one of the earliest systematic literature reviews of traceability approaches (specifically for software architecture and source code) analysed efforts in automatic traceability reconstruction using machine learning classifiers to detect tactic-related classes (Javed and Zdun, 2014): classes that were instrumental to implement the tactical design decisions. This paper studies efforts in NLP application for traceability in recent years, postdating the study done in 2014 (Borg et al., 2014). Our study aims to look into recent applications of NLP by leveraging (natural language) semantics already present in these artifacts. This is an ongoing focus area in the field of information retrieval, particularly due to recent developments in computational power and the advent of large amounts of linguistic data (Torfi et al., 2021). There is a great amount of necessity to consolidate and study these sporadic efforts across different platforms globally to analyse trends in the

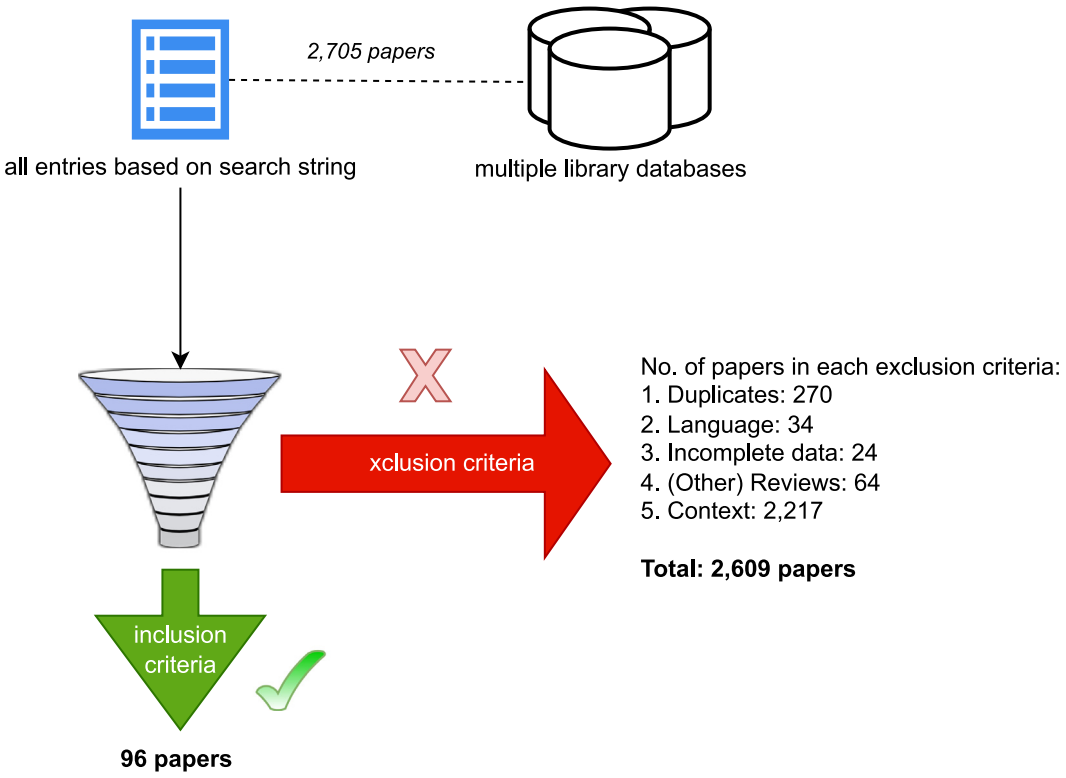


Fig. 1. Overview of steps in mapping study planning.

techniques and tools used, analyse trends of traceability across the SDLC phases, and analyse the open challenges pertaining to NLP application for traceability. Our work contribution progresses in this trajectory, similarly to a checkpoint of reference through analysis of past recent work and recommendations for future efforts.

3. Methodology

Following the updated guidelines for conducting systematic mapping studies in software engineering (Petersen et al., 2015), we define our methodology through the process of identifying, analysing, and interpreting all available evidence in a way that is unbiased and (to a degree) reproducible. The following steps were taken to address our research questions outlined in Section 1.

3.1. Mapping study planning

An overview diagram of our steps is shown in Fig. 1.

We extracted the content and metadata of each piece of literature using a systematic approach and applied various tools to gather all publications necessary within our scope. As shown in Fig. 1, 3.55% of the total result entries have been included as part of our study. This planning was done to ensure comprehensiveness in the study and to address the research questions at hand. Threats to the validity of our study strategy will be discussed in Section 5.

3.2. Search string

Table 1 shows the terms relevant to our search and their synonyms. These were derived to expand the boundaries of semantic keywords that are relevant to the research topics. We have separated the terms according to the relevant theme it belongs to, and only the most relevant synonyms (to our research questions) are shown in the table.

Forming the search string is the core component of any search strategy of a systematic review or mapping study that involves searching indexed literature databases, as it enables transparency for validation and reproducibility for others. An effective search strategy is usually iterative and benefits from trial searches using various combinations of search terms derived from the research question(s) (Kitchenham and Charters, 2007). This was incorporated into our search strategy in our study as follows.

3.2.1. Evaluating synonym terms

Including all the identified synonym terms would yield a wide coverage but be inundated with a great number of false positives. Hence, we evaluate the potential candidate synonym terms to determine those that will be included as our string output. The three components (themes) of our search string will need to be joined using the AND operator, which ensures that results will reflect a “must" rule that all these themes need to be covered. For the individual terms in each theme, we use the OR operator to join them. This is to ensure that every theme is represented by at least one of the terms.

3.2.2. Trial of potential candidate terms

Fig. 2 shows the combination of terms that were tested. We grouped the synonyms according to common properties they share, denoted by the ovals. Each of these groups are then evaluated on effectiveness through trials and decision is then made. Green coloured groups were those chosen.

3.2.3. Decision and final string output

- Theme 1: (top-down order) Main terms, parent term, methods, model types, subject, and artificial intelligence.
- Theme 2: (top-down order) Main terms and offshoot terms.

Table 1
Terms table.

Theme	Term	Synonyms
Natural language Information Retrieval (NL-IR)	NLP, natural language processing	Information retrieval, natural language understanding, text mining, language model, embedding, linguistic, lexical, text extracting, machine learning
Traceability	Traceability	Trace link recovery, trace retrieval
Software artifact	Software artifact	Source code, tests, documentation, requirements

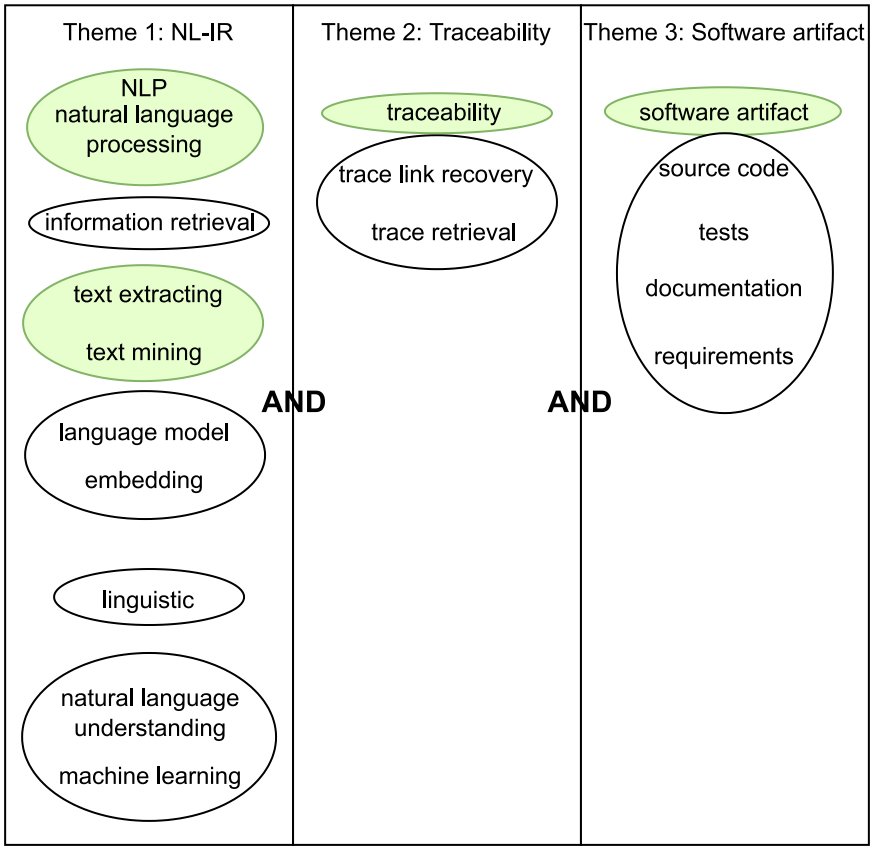


Fig. 2. Grouped synonym terms: potential candidates for search string.

- Theme 3: (top-down order) Main terms and types of artifacts.

For Theme 1, NLP and the meaning of its acronym had to be included. We also found out that the generic term "information retrieval" widened the results beyond the scope of our RQs. The methods group for Theme 1 had to be included because the majority of efforts in text processing do include natural language, although not explicitly mentioned in every case. "linguistic" as a term was producing similar results to "information retrieval" and the artificial intelligence group was not effectively returning the right hits. NLP solutions that already use any form of machine learning is already included when using just the main terms "NLP".

The terms in Theme 2 were more straightforward. We found out that using the term "traceability" was enough to generate the relevant papers in scope of our RQ, as the term is a commonly used term in software engineering, even without including the term "software". We also discovered that lemmatising "traceability" to "trace" and adding "link" was useful to pick up cases where traceability happens without specifically mentioning that it is a traceability problem. For example, locating bugs in the source code, or linking requirements to test cases.

For Theme 3, we found out that including the artifact types into our search string restricted our scope of search — this is particularly due to the nomenclature used to represent artifacts produced throughout the SDLC, which can be numerous. We decided to only use the main terms, both spellings of "artifact" and "artefact". As a result, we specified the following search string (in order) to extract all related publications within our scope:

```
("NLP" OR "natural language processing" OR "text mining" OR "text extracting")
AND
("traceability" OR "trace link")
AND
("source code" OR "software code" OR "software artifacts" OR "software artefacts")
```

As control papers, we used a 10% random sample of the set of papers obtained in the query: for the updated search query, the control papers used were (Pruski et al., 2015; Lin et al., 2021; Khatiwada et al., 2017; Salih et al., 2021; Ali et al., 2018; Lam et al., 2015; Capobianco et al., 2013a; Iammarino et al., 2020; Scanniello et al., 2015). These were analysed by the second author

Table 2
Details of index databases used.

Database	Extraction type	Results
Web of Science	Web UI	9
Springer Link	Web UI	198
ACM Digital Library	Web UI	163
IEEE Xplore	Web UI	18
Scopus	API	27
Google Scholar	API	2290
Total count		2705

to make sure that the search query was appropriate, or if it needed different terms.

3.3. Inclusion and exclusion criteria

To ensure our results are reflective of recent research, we have imposed inclusion criteria in terms of period scope: years 2013 to 2021. Spanning a period of 9 years in consideration, we aim to fill in the gap of studies that predated our start year and focus on more recent developments of NLP-based IR in software traceability. For exclusion, we have disregarded content that is unrelated to (software engineering) traceability, such as other reviews and artifacts with no natural language.

For the exclusion criteria, we used the following filters to weed out the papers that are not within our scope:

1. Duplicates: repeated entries
2. Language: non-English papers
3. Data: incomplete (missing) data
4. Reviews: other reviews, surveys, and mapping studies
5. Context: irrelevance to our defined research topics

The exclusion process (filtering) of papers was necessary due to the abundant false positive results majorly from Google Scholar. Duplicates were identified through automated checking of integrity in titles and authors. For language, we only included those written in English. Incomplete and missing data refers to search results that do not fully reflect published material, for example, only the publication source was mentioned with no article title. We also excluded all other secondary and tertiary studies.

The final filter was 'context'. We had to determine if the papers were *relevant* to our defined research topics. We start with the abstract (as they typically serve as the first point of entry). If relevance is not evident, we look into the research questions and methodology, as these describe the work done to achieve a goal and to answer the research questions. The first author was responsible for this task: 9 control papers (as defined above) were read by the second author to make sure that the context was relevant for the papers to be included. Since the control papers (selected randomly) were all found to be relevant to the context chosen by the research question, a Cohen's kappa was not evaluated as not necessary to determine the agreement between reviewers.

3.4. Data extraction and management

Table 2 shows the literature databases that were used for our first step in data extraction. The aim was to gather all relevant publications related to our study topics, and by using the search string defined. The extraction was done either by exporting from the web page (via manual extraction using the Web UI) or API.

Google Scholar was further used to widen our search results: despite the abundance of false positives (noise), it has the potential to considerably extend the outreach of the systematic search (Harzing and van der Wal, 2008). The results in impact analysis (of publications) will be covered in Section 4.

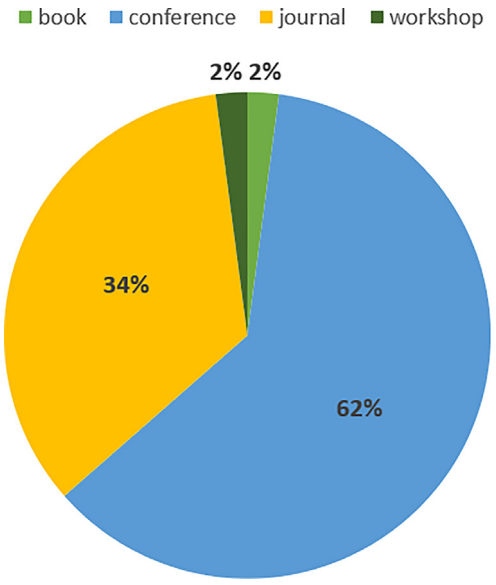


Fig. 3. Distribution of publication types.

After the cleaning step instrumented by the exclusion criteria, we gathered a total of 96 papers held by libraries worldwide. We have also ensured that all these were peer-reviewed publications. These were extracted, along with the metadata, and compiled into a spreadsheet consisting of all the information and content for each paper.

4. Results

The following are the results of our study based on our research topics. These results reflect our findings in NLP efforts in software traceability in recent years, answering our research questions at hand.

4.1. RQ1: Demographics of published articles

In terms of demographics for impact and quality analysis, we look at the following metrics:

- publication type, shown in Fig. 3
- citation count per year,[1] shown in Fig. 4

The complete list of papers in scope can be found in Appendix A. We have also included the respective sources (e.g., conference name) of each paper. The distribution of accepted papers is roughly two-thirds geared towards conference and workshop contributions, and the rest in more established venues (books and journals). This is further proof that conference papers still attract quality contributions, although, as relevant and well-known as a conference might be, this does not define the quality of the papers that are contained in one. Some noticeable conference venues are namely the International Conference of Software Engineering (ICSE, 4 papers), International Conference on Software Maintenance and Evolution (ICSME, 4 papers), and International Requirements Engineering Conference (RE, 4 papers). These are also examples of A*/A rated software engineering conferences,

[1] Number of citations / (current year–published year).

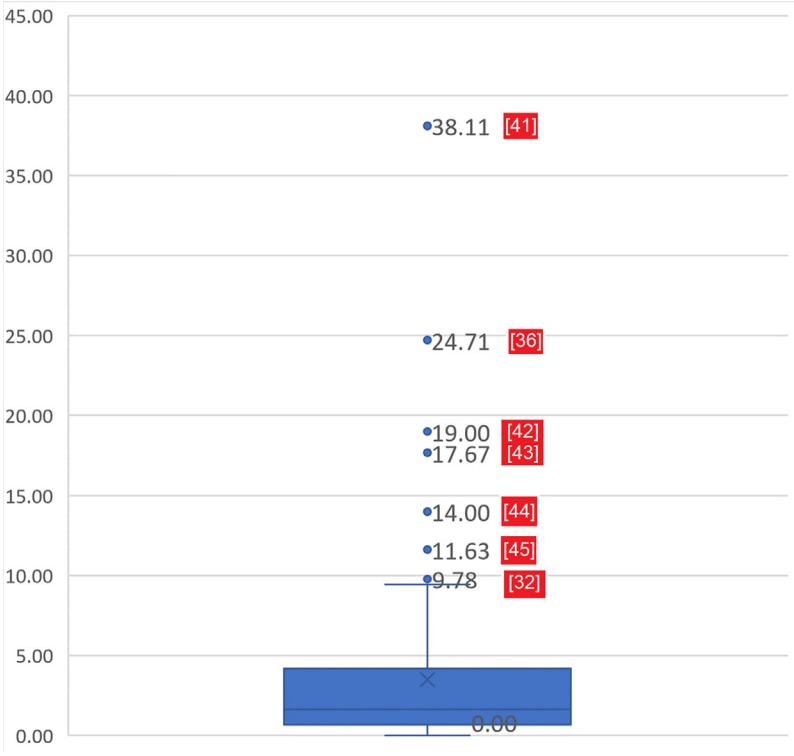


Fig. 4. Box plot of citations per year — outliers are labelled with their respective values.

as listed in CORE conference rankings,[2] which are labelled as flagship and excellent conference venues.

For citation count per year, we can see 7 outliers that are the top cited publications per year, corresponding to the papers (Panichella et al., 2013; Lam et al., 2015; Arora et al., 2015; Shokripour et al., 2013; Poshyvanyk et al., 2013; Wang et al., 2014; Lin et al., 2021). Despite the citation count to be, arguably, a weak indicator of research quality for some (Aksnes et al., 2019), for the purpose of our mapping study, we consider citation count as a factor in research impact, and we will analyse these in Section 5.

4.2. RQ2: Trend analysis of NLP techniques and tools for traceability

In this study, we identify how NLP is being used to achieve traceability solutions. Not all NLP efforts are similar; hence, it is useful to categorise these efforts by amount of task complexity, so we can understand *how much* of NLP was involved in the traceability solutions. We categorise according to the following tiers:

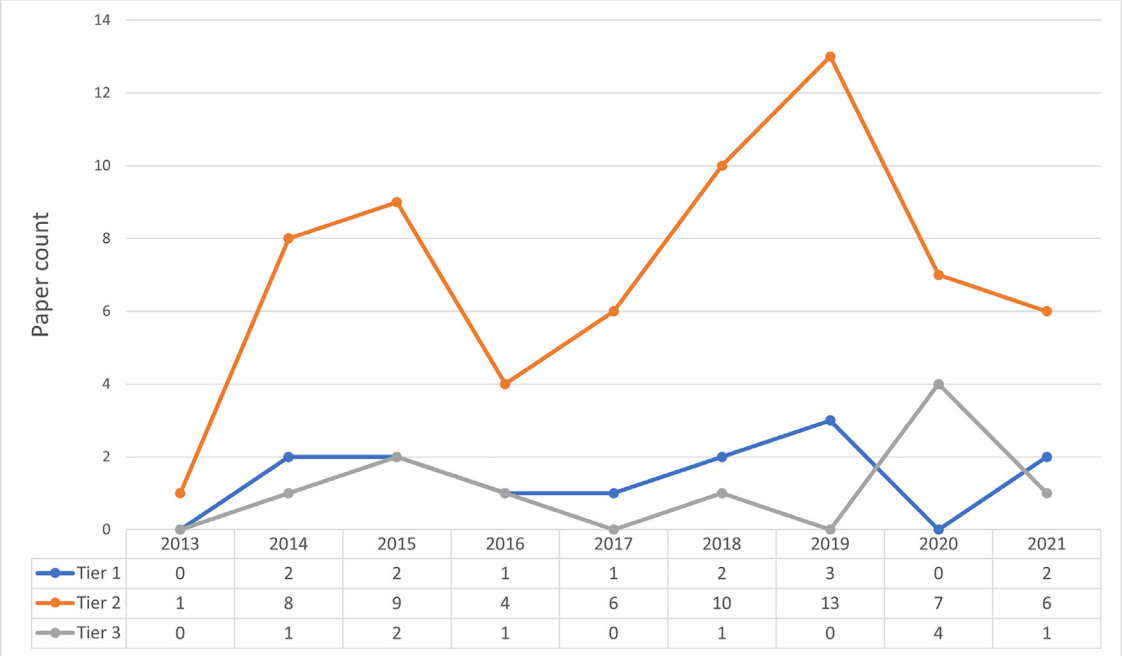
- Tier 1: Only basic complexity tasks, such as processing text (stemming, pattern matching etc.) and tokenising. This category typically only deals with text syntax and no training is involved.
- Tier 2: Basic to intermediate tasks, such as training word embeddings and topic modelling. This category involves training models, pre-trained or otherwise. Semantics are involved and this is closely related to the *naturalness* of language.
- Tier 3: Basic to advanced tasks, such as implementing deep learning models. This category is an extension of Tier 2 where the semantics (context) of language is derived by (essentially) deep learning. This commonly involves the extended implementation of pre-trained deep learning models in the context of software traceability, such as augmenting neural networks with vector space models (VSM).

Distinction between these tiers is solely determined by task complexity: how much work (in processing natural language) has been done (not only for traceability purposes) to achieve the desired solution. For example, traceability work that uses a pre-trained deep learning model (e.g., BERT) would be classified as Tier 3 because deep learning is a relatively high-complexity task albeit being already pre-trained. It is important to note that these tiers are not disjointed, but rather, each tier is an extension of the preceding tier. Tier 2 would include tasks in Tier 1 and Tier 3 would include tasks in Tiers 1 and 2. For example, to train a transformer like BERT (Tier 3), basic work tokenisation still needs to take place. Regardless, segregating these tiers is necessary as it allows us to understand 'up to' what level of task complexity is involved in each paper. Classification of these tiers was performed based on the following steps:

1. In each paper, we extract two sections where present: Introduction and Methodology.
2. In the order listed above, we locate the application of NLP based on the proposed solution. Most of the proposed solutions are explained in the Introduction, although when not clear how NLP is applied, we use the Methodology section to identify the keywords which describe the task complexity involved.
3. Every solution typically involves multiple aspects, and where multiple NLP techniques and tools are applied, only the highest complexity is assigned.

The classification of papers into the three tiers was performed by the first author. The second author, using the subset of publications used as *control papers* above, used the same three steps

[2] http://portal.core.edu.au/conf-ranks



	2013	2014	2015	2016	2017	2018	2019	2020	2021
Tier 1	0	2	2	1	1	2	3	0	2
Tier 2	1	8	9	4	6	10	13	7	6
Tier 3	0	1	2	1	0	1	0	4	1

Fig. 5. Paper count throughout the years.

to determine which tier a paper belongs to. The results of the two classifications were later discussed agreement was sought. Unsurprisingly, there was a 100% agreement between the two authors on this sample of papers: this is due to how the tiers are formulated, and by how clearly each tier is defined from the others.

Based on our analysis, all the task complexity properties are transitive: Tier 1 tasks are a subset of Tier 2 tasks and Tier 2 tasks are a subset of Tier 3 tasks. For example, one does not train a word embedding without having to pre-process the text and one does not train a deep learning model without having to embed layers of word vectorisation models. Thus, these tiers are not disjointed — higher tiers will include tiers that have lower complexity levels, in other words, an implicit "up to" is implied for each. Fig. 5 shows the trend of the tools and techniques involved from a tiered perspective, in terms of published paper count, and by year of publication.

Within each tier, we have multiple techniques and tools that were used as part of traceability solutions in our study. Table 3 shows NLP techniques involved in traceability solutions with the relevant papers involved. Table 4 shows external support tools and libraries that have been identified with the relevant papers involved.

4.3. RQ3: Trend analysis of NLP application for traceability across the SDLC phases

We look into traceability applications through the phases of the SDLC framework. Given that there is not one official SDLC model, we will be using the common *de facto* phases of the framework as our basis (Mishra and Dubey, 2013):

1. REQ: Requirements engineering (problem understanding)
2. DES: Design (planning)
3. CODE: Coding (implementation)
4. TEST: Testing
5. OPS: Deployment & Maintenance

To visualise the relationships identified effectively, we present Fig. 6: a bubble chart of the *pairwise* SDLC phase relationship counts over the years. The horizontal dotted line across 'REQ-CODE' shows the SDLC phase relationship that is present in all years, with 2019 showing the maximum count overall. Where there is no bubble in place, it means that the count is zero.

Every paper in scope has been involved in one or more pairwise SDLC relationships. In cases where papers involve multiple pairwise relationships (which is few), those papers will exist in every bubble, where the pairwise relationship is present for that year. In other words, every paper is not exclusive to every bubble — multiple bubbles may represent one paper that has multiple pairwise relationships. The distribution count is as follows:

- No. of papers with one pairwise relationship: 84
- No. of papers with two pairwise relationships: 11
- No. of papers with three pairwise relationships: 1
- Total no. of papers involved: 96

Fig. 6 also shows the 'OTH' (others) phase, which refers to artifacts involved outside of the SDLC phases identified in Section 4.3. Some examples of artifacts identified at 'OTH' are (informal) documentation, user queries, and release notes.

4.4. RQ4: Key issues, barriers, and setbacks

We have identified eight key issues, barriers, and setbacks, outlined in Table 5, with relevant papers highlighting each of these. These were identified through analysing the discussion of results, which is typically found in the 'Discussion' section of each paper. We extracted all identifiable (implicit or explicit) issues, barriers, and setbacks that are direct results of using NLP in the proposed traceability solutions. Each of these is explained in this section and further discussed in Section 5.

4.4.1. Syntax convention

There does not exist a unified convention for naming syntax of various references in the artifacts, such as functions, variables, and classes. Due to this, we cannot generalise every model

Table 3
NLP techniques identified.

	Paper reference	Technique examples
Tier 1	Arunthavanathan et al. (2016), Wijesinghe et al. (2014), Salih and Sahraoui (2018), Alobaidi and Mahmood (2015), Kchaou et al. (2019), Nishikawa et al. (2015), Salih et al. (2021), Keim and Koziolek (2019), Pruski et al. (2015), Keim et al. (2021), Kchaou. et al. (2017), Zamani et al. (2014), Lin et al. (2017), Pruski et al. (2014), Rasekh et al. (2017), Li and Cleland-Huang (2013), Shokripour et al. (2013)	Parts-of-Speech (POS) tagging, Stemming, Lemmatising, Tokenising, Stopwords removal, Regular expressions, Key phrase extraction, Terms frequency-inverse document frequency (TF-IDF).
Tier 2	Falessi et al. (2016), Kicsi et al. (2021), Zhao et al. (2017b), Kicsi et al. (2018), Csuvik et al. (2019b), Rubasinghe et al. (2018a), Hariri and Fredericks (2018), Lapeña et al. (2017), Rubasinghe et al. (2020), Florez (2019), Singh (2022), Pauzi and Capiluppi (2021), Lapeña et al. (2019), Tian et al. (2018), Hey (2019), Wang et al. (2018), Yıldız et al. (2014), Wang et al. (2019), Rubasinghe et al. (2018b), Liu et al. (2020a), Velasco and Aponte Melo (2019), Ali et al. (2015), Effa Bella et al. (2019), Mahmoud and Bradshaw (2015), Panichella et al. (2015), Csuvik et al. (2019a), Pauzi and Capiluppi (2020), Qusef et al. (2014), Alazzam et al. (2014), Gadelha et al. (2021), Iammarino et al. (2020), Rasekh et al. (2019), Mills and Haiduc (2017), Tsuchiya et al. (2015), Liu et al. (2020b), Hey et al. (2021), Ali et al. (2018), Chen et al. (2019), Divya et al. (2014), Huang et al. (2016), Zhang et al. (2016b), Mahmoud and Niu (2015), Chen et al. (2021), Champagne and Carver (2020), Arora et al. (2015), Heck and Zaidman (2014), Khatiwada et al. (2017), Liu et al. (2019), Effa Bella et al. (2018), Mahmoud and Williams (2016), Mahmoud (2015), Scanniello et al. (2015), Xia et al. (2014), Xie et al. (2019), Wang et al. (2014), Yang and Lee (2021), Malhotra et al. (2018), Zhou et al. (2017), Eder et al. (2015), Zhang et al. (2016a), Gharibi et al. (2018), Capobianco et al. (2013b), Thommazo et al. (2013), Dasgupta et al. (2013), Panichella et al. (2013), Borg et al. (2013), Poshyvanyk et al. (2013), Berta et al. (2017), Zhang et al. (2021)	Latent semantic indexing (LSI), Latent semantic analysis (LSA), Embeddings, Vector space model (VSM), Topic modelling, Translation (language), Named entity recognition (NER), Document/Sentence/Word similarity.
Tier 3	Chen et al. (2018), Malik et al. (2016), Mahmood et al. (2015), Thommazo et al. (2014), Lin et al. (2021), Csuvik et al. (2020), Keim et al. (2020b,a), Lam et al. (2015), Jiang et al. (2020)	Deep learning, Neural net.

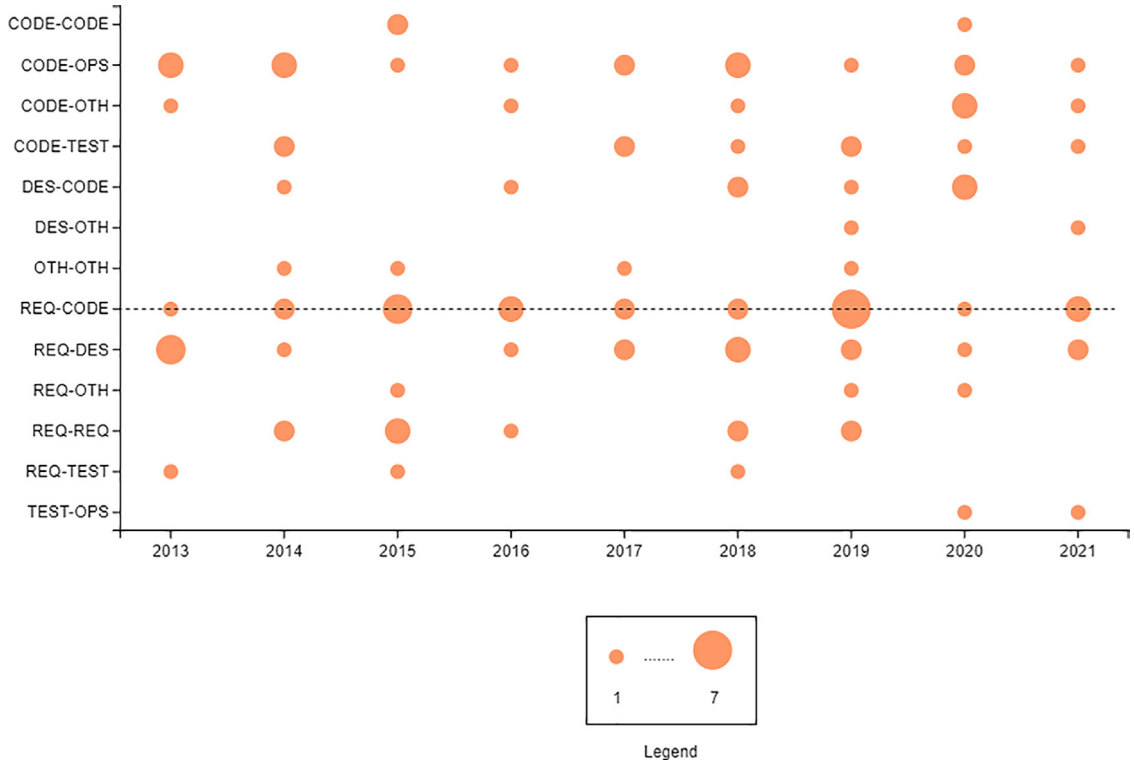


Fig. 6. Bubble chart of SDLC Phase Relationships throughout the years – each bubble size represents the count of papers corresponding to the SDLC phase relationship, shown in the legend for reference.

to be trained on certain specifics, and this hampers effective traceability efforts.

4.4.2. Configuration

Finding the optimum configuration may be possible for one use case. However, in reality, artifacts evolve over time (through active development), and (optimal) configurations change as well. Although NLP has been effective in recovering missing and broken trace links, it is still a pertinent issue in achieving effective traceability. In deep learning tasks (Tier 3), searching for the optimal configuration (exhaustive evaluation) poses other issues, such

Table 4
External NLP supporting tools/libraries identified.

Tools	Paper reference
WordNet[a]	Arunthavanathan et al. (2016), Falessi et al. (2016), Rubasinghe et al. (2018a), Hariri and Fredericks (2018), Rubasinghe et al. (2020), Alobaidi and Mahmood (2015), Kchaou et al. (2019), Malik et al. (2016), Mahmood et al. (2015), Wang et al. (2018), Rubasinghe et al. (2018b), Liu et al. (2020a), Mahmoud and Bradshaw (2015), Rasekh et al. (2019), Liu et al. (2020b), Zhang et al. (2016b), Mahmoud and Niu (2015), Pruski et al. (2015), Kchaou. et al. (2017), Khatiwada et al. (2017), Lin et al. (2017), Liu et al. (2019), Mahmoud and Williams (2016), Pruski et al. (2014), Rasekh et al. (2017), Gharibi et al. (2018), Dasgupta et al. (2013), Berta et al. (2017)
StanfordNLP[b]	Arunthavanathan et al. (2016), Falessi et al. (2016), Rubasinghe et al. (2018a), Hariri and Fredericks (2018), Rubasinghe et al. (2020), Alobaidi and Mahmood (2015), Malik et al. (2016), Mahmood et al. (2015), Wang et al. (2018), Rubasinghe et al. (2018b), Zhang et al. (2016b), Arora et al. (2015), Pruski et al. (2015), Khatiwada et al. (2017), Zamani et al. (2014), Lin et al. (2017), Liu et al. (2019), Mahmoud and Williams (2016), Jiang et al. (2020), Zhou et al. (2017)
Apache Lucene[c]	Zhao et al. (2017b), Alobaidi and Mahmood (2015), Chen et al. (2018), Ali et al. (2015), Alazzam et al. (2014), Mills and Haiduc (2017), Scanniello et al. (2015), Yang and Lee (2021), Zhang et al. (2016a), Gharibi et al. (2018)
ANTLR[d]	Arunthavanathan et al. (2016), Rubasinghe et al. (2018a, 2020, 2018b)
Dbpedia[e]	Alobaidi and Mahmood (2015), Malik et al. (2016), Mahmood et al. (2015)
Babelnet[f]	Alobaidi and Mahmood (2015), Malik et al. (2016), Mahmood et al. (2015), Liu et al. (2020b)
BERT (Devlin et al., 2018)	Kicsi et al. (2021), Thommazo et al. (2014), Lin et al. (2021), Csuvik et al. (2020), Keim et al. (2020b), Hey et al. (2021), Keim et al. (2020a)
NLTK[g]	Falessi et al. (2016), Zhao et al. (2017b), Singh (2022), Wang et al. (2019), Liu et al. (2020a), Gadelha et al. (2021), Hey et al. (2021), Gharibi et al. (2018), Berta et al. (2017)
Gensim[h]	Kicsi et al. (2021, 2018), Csuvik et al. (2019b), Singh (2022), Pauzi and Capiluppi (2021), Wang et al. (2019), Effa Bella et al. (2019), Csuvik et al. (2019a), Pauzi and Capiluppi (2020), Gadelha et al. (2021), Iammarino et al. (2020), Chen et al. (2019), Champagne and Carver (2020), Liu et al. (2019), Effa Bella et al. (2018)
FastText[i]	Pauzi and Capiluppi (2021, 2020), Hey et al. (2021)
SpaCy[j]	Pauzi and Capiluppi (2021, 2020), Gadelha et al. (2021), Hey et al. (2021), Gharibi et al. (2018)
GATE[k]	Malik et al. (2016), Mahmood et al. (2015), Arora et al. (2015), Zamani et al. (2014)
GloVe[l]	Effa Bella et al. (2019), Gadelha et al. (2021), Liu et al. (2019), Gharibi et al. (2018)
Apache OpenNLP[m]	Arunthavanathan et al. (2016), Lapeña et al. (2019), Salih et al. (2021), Mahmoud and Niu (2015), Arora et al. (2015), Mahmoud and Williams (2016)

[a] https://wordnet.princeton.edu
[b] https://nlp.stanford.edu
[c] https://lucene.apache.org
[d] ANother Tool for Language Recognition: https://www.antlr.org
[e] https://www.dbpedia.org
[f] https://babelnet.org
[g] Natural Language ToolKit: https://www.nltk.org
[h] https://radimrehurek.com/gensim
[i] https://fasttext.cc
[j] https://spacy.io
[k] General Architecture for Text Engineering: https://gate.ac.uk
[l] Global Vectors for World Representation: https://nlp.stanford.edu/projects/glove
[m] https://opennlp.apache.org

as computational costs, time complexities, and hardware carbon footprint (Lauriola et al., 2022).

4.4.3. Translation (language)

Translation of languages is a service that is integral to any traceability solution that involves unifying cross-language artifacts. Dependency on the effectiveness of this service (by the accuracy of cross-language information retrieval output) proves to be a setback to effective traceability. A comparative study done in 2015 observed that different translation services can result in considerably different retrieval behaviours for individual queries for different language pairs and applications (Hosseinzadeh Vahid et al., 2015).

4.4.4. Properties (representation) of artifacts

As we implement traceability solutions using NLP (such as similarities in vectors), software artifact properties constantly change and traceability solutions using NLP do not keep up. Besides change management, this issue is also relevant for the representation of software artifacts throughout different SDLC phases. For example, in the Design phase where UML diagrams are used, some form of parser needs to be implemented to unify these representations with other artifacts from other SDLC phases.

4.4.5. Explainability

The lack of explainable and interpretable models is a key barrier to effective traceability. This becomes more prominent in higher tiers of task complexity as state-of-the-art pre-trained

Table 5
Papers highlighting key issues, barriers, and setbacks.

Issue, Barrier or Setback	Paper reference
1. Syntax convention	Kicsi et al. (2018), Csuvik et al. (2019b), Florez (2019), Kchaou et al. (2019), Rubasinghe et al. (2018b), Keim et al. (2020b), Mahmoud and Williams (2016), Mahmoud (2015), Keim et al. (2020a)
2. Configuration	Hariri and Fredericks (2018), Singh (2022), Ali et al. (2015), Eder et al. (2015), Panichella et al. (2013)
3. Translation (language)	Yıldız et al. (2014), Liu et al. (2020a), Xia et al. (2014)
4. Properties (representation) of artifacts	Florez (2019), Wang et al. (2018), Effa Bella et al. (2019), Panichella et al. (2015), Csuvik et al. (2019a), Pauzi and Capiluppi (2020), Huang et al. (2016), Mahmoud and Niu (2015), Arora et al. (2015), Khatiwada et al. (2017)
5. Explainability	Arunthavanathan et al. (2016), Velasco and Aponte Melo (2019)
6. Dependency on tacit knowledge	Lapeña et al. (2019), Keim and Koziolek (2019)
7. Scalability	Rubasinghe et al. (2020), Chen et al. (2018), Velasco and Aponte Melo (2019), Mahmoud and Bradshaw (2015), Tsuchiya et al. (2015)
8. Data availability	Chen et al. (2021)

models, although scoring high in benchmarked NLP tasks, are typically black-box in nature and serve very little purpose in situations where traceability becomes a core component mandated by requirement standards and regulations, such as for medical device software (Regan et al., 2013).

4.4.6. Dependency on tacit knowledge

There is still a considerable amount of dependency on tacit knowledge that is integral to traceability solutions with NLP. This dependency is hampering efforts in automated effective traceability due to the limitations of models in every domain, which is also related to the artifacts property (representation) issue where it is not a one-size-fits-all policy for all SLDC phases.

4.4.7. Scalability

Scaling the solutions in traceability efforts is identified as a key barrier, particularly in large-scale systems. In object-oriented programming, encapsulation of objects helps to improve scalability due to the isolation of internal modifications of any one object (Corriveau, 1996). Despite this, traceability between software artifacts does not automatically follow this, especially when large systems involve complex trace links with the increasing number of artifacts and developers involved. This is also an extension to the configuration issue where scalability in compute and time complexities are severely affecting effective traceability efforts.

4.4.8. Data availability

In supervised and semi-supervised strategies, we require vast amounts of training data specific to the software engineering domain. In an ideal world, all of this data is annotated and ontologies are well-defined; however, that is not the case in reality. Annotation of data is an expensive and time-consuming laborious task that does not appeal to many – and this has prompted a variety of solutions such as crowdfunding through Amazon Mechanical Turk (Snow et al., 2008).

4.5. RQ5: Open challenges

From these key issues, barriers and setbacks, we identify 3 themes that are presented as *open challenges* in recent applications of NLP in traceability.

4.5.1. Syntax and semantic similarities in representation across artifacts

Traceability between artifacts stems from identifying components that are linked to one another. To achieve this, the manifestation of concepts (through the artifacts' components) needs to be synchronised in terms of syntax and semantic similarities. This challenge is one that NLP solutions for traceability continue to face.

4.5.2. Effectiveness in automated software traceability

As software systems continue to evolve in scale and complexity, the call for automated traceability has never been more critical. The number of traceability links that need to be captured exponentially grows with the size and complexity of the software system (Cleland-Huang et al., 2003). Moreover, consistent changes throughout the SDLC pose a significant challenge to the maintenance of traceability links, with studies showing that change can be expected throughout the life cycle of every project (Boehm, 2003). In the noble quest for automated traceability, the effectiveness of these solutions continues to be an open challenge.

4.5.3. Achieving scalable, adaptive, and explainable models

Recent works (especially in deep learning and off-the-shelf solutions) have resulted in an increasing number of black-box NLP services and tools. Traceability solutions need to be transparent, especially when traceability is a factor in requirements validation and tracing of regulations. Moreover, the challenge of scaling and adapting NLP solutions continues to be an open challenge for interoperability. Any trade-offs between implementing an NLP component to achieve successful traceability, and the extra resource it needs, have to be justified.

5. Discussion

To further elaborate our findings based on our research questions outlined in Section 1, we will discuss the results of our study.

5.1. RQ1: Demographics and quality analysis

Fig. 3 shows the percentage spread of publication type, with conference proceedings (62%) and journal articles (34%) making up most of the papers selected. All of the conferences and journals (where the papers selected were published) were peer-reviewed and some were shown as outliers for having higher citation per year metrics compared to the dataset (Fig. 4).

In Computer Science, the citation count of conferences is no higher than in journals. Moreover, analysis has shown that Computer Science, as a discipline, values conferences as a publication venue more highly than any other academic field of study (Vrettas and Sanderson, 2015). As we look into our outliers more closely, we present a summary of the traceability solutions proposed in each and how NLP was applied – shown at Table 6 (only those with cites per year $\geq$ 10 are shown). As visible in the table, the majority of the outlier papers come from the top publishing venues in software engineering (ACM/IEEE International Conference on Software Engineering and IEEE Transactions on Software Engineering) and the citations reflect a growing trend as long as the paper gets older.

Table 6
Top cited papers per year identified as outliers.

Citations per year	Paper title & reference	Publication source	Summary of NLP application for traceability
38.11	How to effectively use topic models for software engineering tasks? an approach based on genetic algorithms (Panichella et al., 2013)	2013 35th International Conference on Software Engineering (ICSE)	LDA-GA: Using Genetic Algorithms (GA) to determine near optimal configuration for LDA topic modelling.
24.71	Combining deep learning with information retrieval to localise buggy files for bug reports (n) (Lam et al., 2015)	2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)	HyLoc: Combining Deep Neural Network (DNN) with rVSM (revised Vector Space Model) for bug localisation.
19.00	Automated checking of conformance to requirements templates using natural language processing (Arora et al., 2015)	IEEE Transactions on Software Engineering	Template Conformance Checking (TCC): Text chunking and pattern matching to automate requirements conformance.
17.67	Why so complicated? simple term filtering and weighting for location-based bug report assignment recommendation (Shokripour et al., 2013)	2013 10th Working Conference on Mining Software Repositories (MSR)	Two phase location-based approach to bug localisation by predicting relevant files and creating a noun index.
14.00	Concept location using formal concept analysis and information retrieval (Poshyvanyk et al., 2013)	ACM Transactions on Software Engineering and Methodology	Using LSI to map textual descriptions of features or bugs to source code.
11.63	Compositional vector space models for improved bug localisation (Wang et al., 2014)	2014 IEEE International Conference on Software Maintenance and Evolution (ICSME)	Composing various VSM variants based on Genetic Algorithms (GA) for bug localisation.
10.00	Traceability transformed: Generating more accurate links with pre-trained bert models (Lin et al., 2021)	2021 43rd International Conference on Software Engineering (ICSE)	Trace BERT (T-BERT): Three step training of T-BERT models to recover links between issues and commits.

Table 7
Citation analysis per tier.

Category	Total paper count	Average citations per year
Tier 1	15	2.68
Tier 2	71	3.52
Tier 3	10	4.51

5.2. RQ2: Trend analysis of NLP techniques and tools for traceability

We look into how the techniques and tools in NLP evolved over the past recent years. Based on Fig. 5, we can see that the majority of NLP efforts are in the Tier 2 category: involving 'basic' to 'intermediate' tasks, with a prominent spike in 2019. During the early years of our scope (2013–2017), these were used mainly to process text and represent text into vectors, and using the represented vectors in a space model (VSM etc.) to detect similarities. The role of NLP has evolved over recent years due to the proliferation of efforts in combining machine learning with basic text processing. This trend continues, with a focus on deep learning, such as with transformers (Vaswani et al., 2017). The spike in 2020 (for Tier 3) may be attributed to the increasing research interest in state-of-the-art deep learning tools in NLP recently, such as the introduction of Convolutional Neural Networks more commonly (prior) used in Computer Vision (Moreno Lopez and Kalita, 2017), BERT (Devlin et al., 2018), and Huggingface Transformers in 2019 (Wolf et al., 2019).

To further understand the trend beyond using the period of years as our timeline, we should consider the research impact that each tier has *(Which areas are being mostly cited? Where is the attention drawing to?)*. This can be done by using citation analysis for each tier; citation per year (for each tier) indicates the amount of attention (impact) the research has. Table 7 shows the average citations per count of each tier category.[3]

From the table, we can see that despite Tier 3 having the least amount of papers published overall, the average citation count per year is the highest of all tiers (4.51). The aforementioned spike in 2020 for Tier 3 is still considerably lower than Tier 2's spike in 2019; however, this citation analysis may indicate that the research impact in deep learning (for NLP applications in traceability) is the largest. It is still too early to conclude how the trend of deep learning in NLP will go (in the field of traceability), but in general, we can see an upward trend in deep learning across software engineering (Ferreira et al., 2021).

5.3. RQ3: Trend analysis of NLP applications for traceability across SDLC

Based on Fig. 6, we can see the SDLC phases where traceability with NLP occurs more frequently, i.e., relationships involving REQ, CODE, and DES phases. As noticed above, Requirements Engineering is the area with the most traceability activities throughout recent years, followed by Design and Bug Localisation, respectively.

5.3.1. Requirements traceability

The trend of tracing requirements to source code (and vice versa) using NLP is very common throughout the years with a considerable spike in 2019, as seen in Fig. 6. Artifacts pertaining to the REQ phase (such as functional and non-functional requirements) are generally written in natural language. There is no observable unified structure behind the language and syntax. Bi-directional traceability (Salih et al., 2021), linking to UML diagrams (Arunthavanathan et al., 2016; Salih and Sahraoui, 2018; Kchaou et al., 2019; Salih et al., 2021; Panichella et al., 2015; Kchaou. et al., 2017; Effa Bella et al., 2018), fuzzy logic (Thommazo et al., 2013), reducing false positives (Effa Bella et al., 2019; Capobianco et al., 2013b), are some examples of how NLP was used during the REQ phase.

Tracing requirements to other artifacts, such as UML diagrams and source code, is necessary, and in some cases, mandatory, to adhere to regulatory compliance. For healthcare systems, we have HIPAA (Healthcare Insurance Portability and Accountability Act) (Florez, 2019; Velasco and Aponte Melo, 2019; Lin et al., 2017; Effa Bella et al., 2018). In airspace systems, National Aeronautics and Space Administration (NASA) strives to ensure FAA

[3] Average citations per year = sum of all citation counts/number of papers.

(Federal Aviation Administration) governance policies and standards are met (Malik et al., 2016). Templates facilitate good quality (inherently an effective tool for conformance) by avoiding complex structures, ambiguity, and inconsistency in requirements. However, managing this conformance is labour intensive and automated checking of conformance to template tool was developed: REquirements Template Analyzer (RETA) (Arora et al., 2015). In some cases, these regulations are explicitly written as non-functional requirements, such as corresponding to safety and legal aspects (Mahmoud and Williams, 2016; Mahmoud, 2015).

5.3.2. Bug localisation

In our study, bug localisation was also a major highlight in several papers across different phases of the SDLC. NLP was used to reduce manual efforts in remedying faults outlined in bug reports by automating redundant tasks such as reading and searching in natural language artifacts, and locating areas of concern. Examples include comparing bugs to generated patches (Csuvik et al., 2020), between bug reports and test cases (Gadelha et al., 2021), between bug reports and source code (Khatiwada et al., 2017; Liu et al., 2019; Wang et al., 2014; Lam et al., 2015; Malhotra et al., 2018; Jiang et al., 2020; Zhou et al., 2017; Gharibi et al., 2018; Shokripour et al., 2013), cross-language bug tracing (Xia et al., 2014), and commit information (Yang and Lee, 2021).

In the current landscape of large evolving software systems, locating bugs (typically within the source code) is a challenging task. Our study looks into traceability between artifacts, and for bug localisation, we have identified bug reports to be the focal artifact involved in bug localisation. Natural language in bug reports is a common target for NLP tasks (such as traceability, which is the entirety of our study), so de-noising these bug reports to isolate the non-natural languages helps the cause (Hirsch and Hofer, 2022).

One common example of bug localisation is *tracing* the components of a bug report to source code. Bug reports are a form of change request, which serves to change the existing program elements (e.g. source code files) to correct an undesired behaviour of the software (Dilshener et al., 2017). This allows developers to identify what needs to be rectified and modified in the source code to remove the bug, which is a core software maintenance task. Through the lens of traceability using NLP, these components may relate to terms that match between bug reports and source code. Empirical studies have shown that vocabulary used in bug reports was also present in the source code files (Moreno et al., 2013; Saha et al., 2013) — be it an exact or partial match of program elements (i.e. class, method, or variable names and comments). This matching (syntax and semantic similarity) paves a way for NLP to determine bug location more effectively.

5.3.3. Continuous developed tools

We have also identified some tools that were developed continuously across the years (covered by multiple papers reflecting incremental development) across the SDLC phases, namely Software Artifacts Traceability Analyzer (SAT-Analyzer) and TiQi. NLP was first introduced in SAT-Analyzer for addressing artifact inconsistencies due to natural language representation (Arunthavanathan et al., 2016) — it improves the usability of SAT-Analyzer through automated generation of XML input from requirement artifacts, which was then evaluated by a case study on a Point-of-sale (POS) system (Rubasinghe et al., 2018b). SAT-Analyzer was also covered in DevOps practices (Rubasinghe et al., 2018a, 2020); a traceability management tool for continuous integration and multi-user collaboration. TiQi, on the other hand, focuses on trace queries that are generally complex and naturally worded, transforming them into executable SQL statements (Pruski et al., 2014). A more in-depth description of the architecture, design, and heuristic rules was then published in a later paper (Pruski et al., 2015), and a demo was made available online (Lin et al., 2017).

5.4. RQ4: Key issues, barriers, and setbacks

We dive into each of these key points to understand further how the papers have contributed to the aforementioned issues, barriers, and setbacks.

5.4.1. Syntax convention

Our study has found that some assumptions had to be made in the semantic representation of syntax used in artifacts. For example, developers only use expressive, non-abbreviated variable names, such as those that are contained in BERT's dictionary (Keim et al., 2020b,a).

Lack of generally used annotation of artifacts (Kicsi et al., 2018) and imperfectly appropriate naming (Csuvik et al., 2019b) typically lead to inaccurate links. The added challenge of artifacts, such as non-functional requirements (Mahmoud and Williams, 2016), hinders traceability efforts due to the lack of homogeneity in syntax representation: natural language pertaining to non-functional requirements is less explicit in tracing links. Moreover, the detection of constraints in non-functional requirements becomes more difficult due to the lack of robust modelling and documentation techniques (Mahmoud, 2015).

In a case study for SAT-Analyzer, it was observed that the inaccurate artifact elements extraction and identification with NLP that contain different naming conventions and less meaningful names in requirement artifacts, have led to the lack of accuracy (Rubasinghe et al., 2018b). Semantic ambiguities in artifacts written in natural language pose a challenge in tracing explicit links with other artifacts, based on the syntax used (Kchaou et al., 2019).

In specific critical contexts, such as healthcare regulations, desired levels of granularity in traceability are often not enough. The regulations related to audit control standards and session expiration in the implementation of healthcare systems were the hardest to trace to source code statements — very few lines and source code structures related to these requirements were successfully mapped (Florez, 2019).

5.4.2. Configuration

Although NLP has been effective in recovering missing and broken links in self-adaptive systems, it can introduce significant overhead (Hariri and Fredericks, 2018). Threshold values of semantic similarity are typically a 'moving goalpost' and high confidence values, such as 95% (Singh, 2022), were chosen arbitrarily to represent strong confidence. Selection and tuning of parameters are an impact factor for the accuracy of results, and static configurations are identified as an internal threat to the validity of results (Ali et al., 2015). Automated configurations, such as for Latent Semantic Indexing (Eder et al., 2015), improve the applicability, although computation overhead can be significant.

As mentioned in the previous section, exhaustive evaluation for optimal configuration results in various complications, such as significant computational costs and time complexities. This is exacerbated by the continuously changing nature of artifacts throughout the SDLC phases, rendering traceability efforts to become even more challenging. Achieving this (near) optimal configuration for topic modelling was the goal of one of our papers, which introduced Genetic Algorithms (GA) with LDA to boost accuracy of traceability link recovery (Panichella et al., 2013), among other tasks. This paper also highlighted the need for an efficient method to find the best configuration of parameters, as an exhaustive analysis of all possible combinations is deemed impractical.

Effective traceability is crucially dependent on the performance of the models used, which is determined by their configuration settings. One key aspect of this is the hyperparameter tuning, which often can make the difference between a mediocre performing model to a state-of-the-art (Eggensperger et al., 2015).

5.4.3. Translation (language)

Reported setbacks in these efforts concern the effectiveness of translation services that are readily available (Yıldız et al., 2014; Liu et al., 2020a; Xia et al., 2014). Despite these translation services being mainly black-box in nature, it is critical to the effectiveness of traceability. There is no generic dictionary (model) for all languages, as each language has its own rules of grammar (syntax) and its own semantic interpretation of words used. However, we do have a recent primer publication on pre-trained multilingual embeddings (Doddapaneni et al., 2021), yet to be fully utilised in software engineering.

5.4.4. Properties (representation) of artifacts

In a dynamic continuously integrated, continuously developing environment (Rubasinghe et al., 2018a, 2020, 2018b), artifacts transform constantly and this hampers continuous traceability efforts. In cases where traceability is necessary for regulations (Florez, 2019; Arora et al., 2015), the natural language used in these documents is not represented similarly to other artifacts, such as functional and non-functional requirements. Adaptive standard feedback was also proposed upon the consideration that software artifacts do not share the same properties of natural language documents, on which the standard feedback relies (Panichella et al., 2015).

Semantic similarities can also be challenging with natural language due to polysemy (Wang et al., 2018), non-uniform identifiers (Pauzi and Capiluppi, 2020), ambiguity in content (Kchaou et al., 2019; Pruski et al., 2014), and vocabulary mismatch (Khatiwada et al., 2017).

5.4.5. Explainability

Despite huge successes in large language models, their black-box nature hinders key goals of NLP, particularly in explainability (Lin et al., 2021; Keim et al., 2020b,a). In cases where traceability plays an important role (such as adherence to regulations and auditing), the black-box nature of these advanced solutions proves as a hindrance, as validation of results becomes difficult (Velasco and Aponte Melo, 2019).

5.4.6. Dependency on tacit knowledge

This is more prominent in traceability use cases pertaining to software architecture where experiential knowledge is vital in recovering architectural trace links (Keim and Koziolek, 2019) and links between requirements and process models (Lapeña et al., 2019).

5.4.7. Scalability

Large-scale systems pose a challenge in traceability management due to the complexity of trace links, particularly in visualisation (Rubasinghe et al., 2020; Chen et al., 2018). This also relates to time and compute resource complexities, and becomes even more challenging in environments where constant change is present (Rubasinghe et al., 2018a,b).

5.4.8. Data availability

The amount of labelled data to train classifiers is not as abundant as we ideally need it to be, and this poses a setback for effective training in supervised models for traceability (Chen et al., 2021). The amount of annotated data in some domains is richer than in others, which is heavily dependent on the efforts of the community. This translates to varying levels of model accuracy for different domains, which affects traceability effectiveness. Models can only train on data that is available, and the performance of any model is entirely dependent on the data that it is trained on.

5.5. RQ5: Open challenges

To answer RQ5, we first need to be able to identify the pertinent issues that arise; and second, through understanding the pain points, we can derive and model the open challenges. Fig. 7 shows the mapping of open challenges from the key issues, barriers, and setbacks that were identified in Section 5.4.

5.5.1. Syntax and semantic similarities in representation across artifacts

The first and foremost open challenge of NLP is primarily derived from the most recurring issue reported in our study (see Section 4.4.4), and centred around the role NLP plays in traceability: processing natural language in artifacts. The natural language present in artifacts needs to be represented uniformly in various parts of the SDLC, and achieving similarity in each of those representations is an open challenge that NLP continues to play a major part in solving.

5.5.2. Effectiveness in automated software traceability

Software artifacts are not entirely similar to that of natural language, and NLP advancement efforts are majorly based on use cases pertaining to human communication, such as developing cognitive (intelligent) skills through natural language understanding. This direction is not entirely useful for software engineering purposes, particularly relating to traceability. The open challenge is in leveraging and harnessing the value of NLP techniques, focusing NLP advancement efforts in the field of software engineering. Moreover, pure automation of traceability efforts continues to pose a common challenge despite recent successes in language models.

5.5.3. Achieving scalable, adaptive, and explainable models

NLP models that are involved in traceability efforts face significant challenges to scale and adapt in tandem with how software systems change and evolve throughout the SDLC. This open challenge is a derivative of identified issues pertaining to scalability, data availability, and explainability. Explainable AI is a critical component to adopting machine learning models in any decision making process, with traceability being no different. In software engineering, the adoption of these models are hindered by the lack of explainability and understanding of how these models work (Tantithamthavorn and Jiarpakdee, 2021).

5.6. Recommendations

Fig. 8 presents a mapping diagram to show the relationships between the open challenges and recommendations. The following are our points of recommendation in addressing the three open challenges, as described above in Section 5.5.

5.6.1. A holistic framework model for NLP solutions to achieve effective traceability

NLP techniques and tools have played a major role in processing and vectorising text; serving as some form of *natural language decoder* to unify representations across artifacts for traceability. We recommend efforts in developing a holistic framework model to achieve effective traceability, subsequently addressing key open challenges of NLP in traceability. A holistic framework should fulfil the following:

- Techniques and tools in NLP that are representative of the software engineering domain. Currently, efforts are sparse and scattered, focusing on very specific parts of software engineering that are isolated.

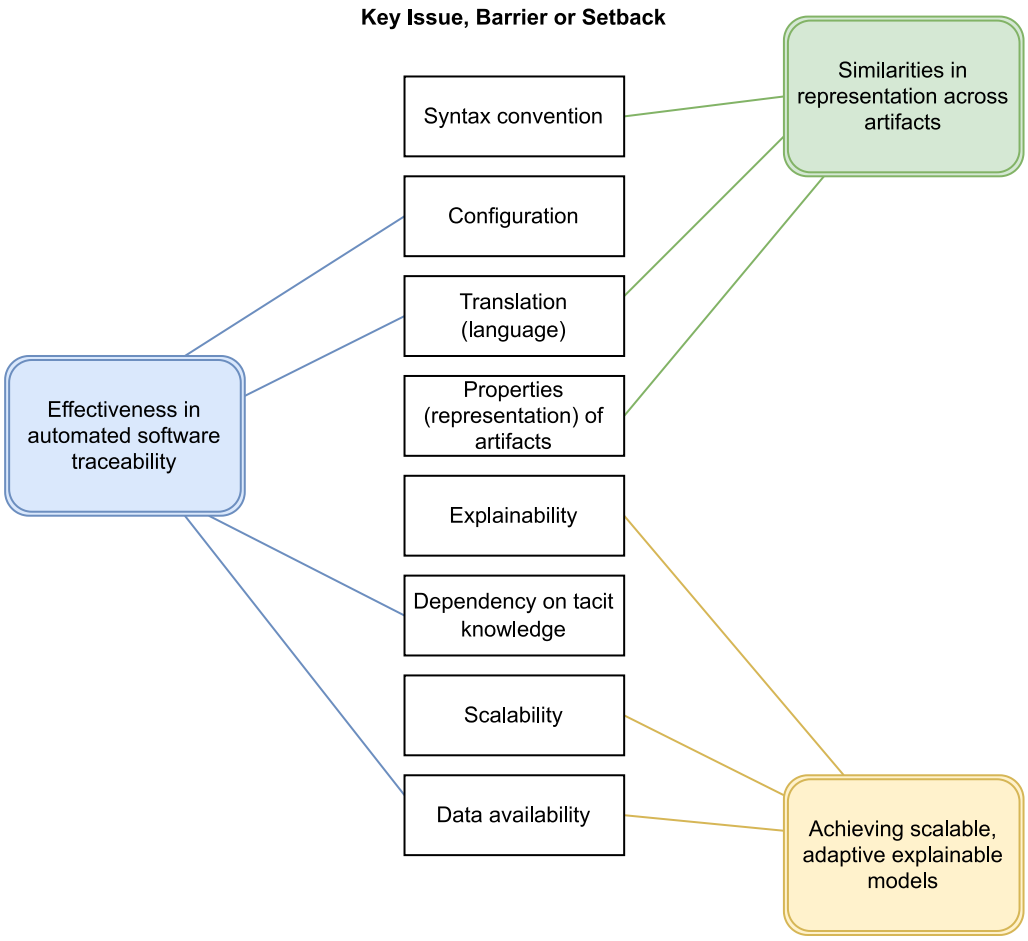


Fig. 7. Mapping of key issues, barriers, and setbacks to open challenges.

- A unified ontology across the software engineering domain space, through consolidating and integrating taxonomies across multiple domains in software engineering.
- Models that 'understand' natural language across various aspects of the SDLC phases. Natural Language Understanding (NLU) is an extension of NLP where models are able to comprehend terms that are specific to the SDLC phases, and across these phases, through classifying intents, confidence scores stability, and extracting entities (Abdellatif et al., 2021).

5.6.2. Towards achieving interoperability and explainability

Models have to be transparent, scalable, and accurate in recovering trace links (i.e. effective traceability). We propose to ensure applications of NLP in traceability to be transparent and explainable. Efforts in NLP research for traceability should not only focus on having the next best model that supersedes the accuracy scores of previous models in determining trace links, but also on proving scalability and providing explainability. We need to have some form of global certification and validation process to be able to certify models as experts. Moreover, we need to incorporate efforts in explainable Artificial Intelligence (AI) and model reasoning to reduce bias and fill in the gap of dependencies on tacit knowledge from human experts; dependencies on experiential knowledge.

6. Threats to validity

In this section, we outline the threats to validity identified throughout our mapping study process. Based on a recent map of threats to validity in systematic mapping studies in software engineering (Zhou et al., 2016), we looked into all possible threats that emerge from conducting our study.

6.1. Construct validity

Our research questions and methodology may not entirely cover every aspect of studying how NLP is used for software traceability. However, we ensured that our research strategy was thorough and comprehensive in fulfilling the secondary study conducted to address key gaps of areas pertaining to NLP in software traceability. We adhered to the guidelines outlined in Petersen et al. (2015). It is important to stress again that a systematic literature review would be less significant to uncover the existing methods and approaches based on NLP, and it would face a larger threat to construct validity than the mapping study presented in this work.

6.2. Internal validity

The search for relevant papers to populate our mapping study was thoroughly executed: multiple library databases were used, including a search aggregate engine that covers a wide range of multiple databases and libraries — Google Scholar engine. Addressing internal threats to validity is critical in mapping studies: the findings need to be unbiased and the search string needs to be reflective of our study scope.

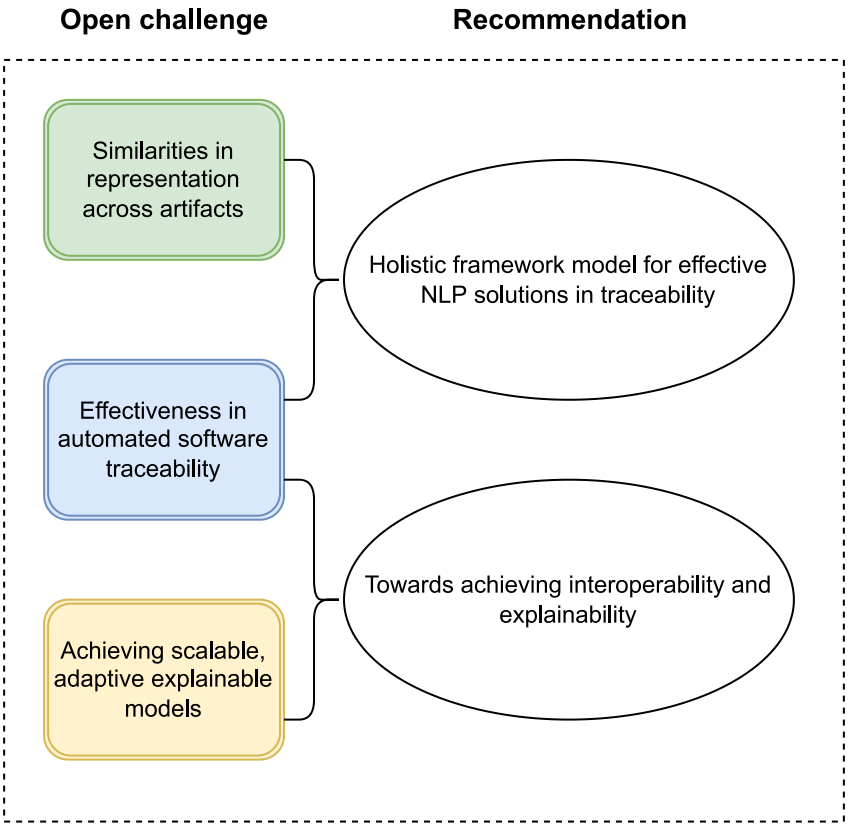


Fig. 8. Mapping of open challenges to recommendations.

6.3. External validity

The specificity of the techniques and tools and trends analysed in our study may not be able to be generalised outside of our search scope. Research efforts in NLP and traceability continue to evolve rapidly in recent years, and focus choice may affect the results generated. In reducing this threat, and for the sake of generalisability, we proposed tier categorisation for NLP techniques and focused our recommendations on common key issues, barriers, and setbacks rather than specific ones.

6.4. Conclusion validity

The limited availability of published efforts in NLP and software traceability may impact the conclusions derived from our study scope, especially on empirical evidence in the industry for traceability efforts that are not published. Incorporating synonyms of terms using the Google Scholar search engine as part of our data ingestion pipeline helped us reduce this threat, despite returning abundant false positives.

7. Conclusion

This paper presents a systematic mapping study focusing on NLP and its applications, in the context of software traceability. A total of 96 papers were obtained – covering a period of years 2013 to 2021 – during the selection process. We looked into the different ways NLP was leveraged to aid traceability efforts across the various phases of the SDLC. We analysed the trend of techniques and tools used, the trend of traceability activities that were involved, and identified key issues, barriers, and setbacks to these traceability efforts. From these, we identified open challenges and presented key recommendations for addressing these.

The field of research in NLP is continuously evolving, and while major use cases of these efforts are typically related to human communication (i.e. human language), there is great potential value for NLP to be further leveraged effectively in software traceability. By conducting this mapping study, we are able to consolidate recent efforts in attempting to take advantage of these techniques and tools to solve traceability problems, particularly through automating redundant tasks and solving key issues that arise from conventional IR techniques. This study serves as a checkpoint for researchers and practitioners to have a wide angle of view across the various efforts within our scope of the study. Based on the trend analysis done and the open challenges identified, this study has presented two key recommendations in moving forward: a holistic framework for NLP solutions and efforts in achieving interoperability and explainability in NLP models.

CRediT authorship contribution statement

Zaki Pauzi: Conceptualization, Methodology, Software, Investigation, Data curation, Writing – original draft, Writing – review & editing, Visualization. **Andrea Capiluppi:** Validation, Resources, Supervision, Writing – review & editing, Project administration.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.jss.2023.111616.

References

Abdellatif, A., Badran, K., Costa, D., Shihab, E., 2021. A comparison of natural language understanding platforms for Chatbots in software engineering. IEEE Trans. Softw. Eng. PP, 1. http://dx.doi.org/10.1109/TSE.2021.3078384.

Aksnes, D.W., Langfeldt, L., Wouters, P., 2019. Citations, citation indicators, and research quality: An overview of basic concepts and theories. SAGE Open 9 (1), 2158244019829575. http://dx.doi.org/10.1177/2158244019829575.

Alazzam, I., Alsmadi, I., Akour, M., 2014. Test cases selection based on source code features extraction. Int. J. Softw. Eng. Appl. 8, 203–214. http://dx.doi.org/10.14257/ijseia.2014.8.1.18.

Ali, N., Cai, H., Hamou-Lhadj, A., Hassine, J., 2018. Exploiting parts-of-speech for effective automated requirements traceability. Inf. Softw. Technol. 106, http://dx.doi.org/10.1016/j.infsof.2018.09.009.

Ali, N., Sharafi, Z., Guéhéneuc, Y.-G., Antoniol, G., 2015. An empirical study on the importance of source code entities for requirements traceability. Empirical Softw. Engg. 20 (2), 442–478. http://dx.doi.org/10.1007/s10664-014-9315-y.

Alobaidi, M., Mahmood, K., 2015. Semantic approach for traceability link recovery using uniform resource identifier (STURI). In: The 2015 International Conference on Software Engineering Research and Practice. CSREA Press, Athens, pp. 190–195, URL http://worldcomp-proceedings.com/proc/p2015/SER2587.pdf.

Arora, C., Sabetzadeh, M., Briand, L., Zimmer, F., 2015. Automated checking of conformance to requirements templates using natural language processing. IEEE Trans. Softw. Eng. 41 (10), 944–968. http://dx.doi.org/10.1109/TSE.2015.2428709.

Arunthavanathan, A., Shanmugathasan, S., Ratnavel, S., Thiyagarajah, V., Perera, I., Meedeniya, D., Balasubramaniam, D., 2016. Support for traceability management of software artefacts using natural language processing. In: 2016 Moratuwa Engineering Research Conference. MERCon, Institute of Electrical and Electronics Engineers (IEEE), Moratuwa, Sri Lanka, pp. 18–23. http://dx.doi.org/10.1109/MERCon.2016.7480109.

Berta, P., Bystrickỳ, M., Krempaskỳ, M., Vranić, V., 2017. Employing issues and commits for in-code sentence based use case identification and remodularization. In: Proceedings of the Fifth European Conference on the Engineering of Computer-Based Systems. pp. 1–8.

Boehm, B., 2003. Value-based software engineering: Reinventing. SIGSOFT Softw. Eng. Notes 28 (2), 3. http://dx.doi.org/10.1145/638750.638775.

Borg, M., Pfahl, D., Runeson, P., 2013. Analyzing networks of issue reports. In: 2013 17th European Conference on Software Maintenance and Reengineering. IEEE, Genova, Italy, pp. 79–88. http://dx.doi.org/10.1109/CSMR.2013.18.

Borg, M., Runeson, P., Ardö, A., 2014. Recovering from a decade: A systematic mapping of information retrieval approaches to software traceability. Empirical Softw. Engg. 19 (6), 1565–1616. http://dx.doi.org/10.1007/s10664-013-9255-y.

Capobianco, G., Lucia, A.D., Oliveto, R., Panichella, A., Panichella, S., 2013a. Improving IR-based traceability recovery via noun-based indexing of software artifacts. J. Softw. Evol. Process 25 (7), 743–762.

Capobianco, G., Lucia, A.D., Oliveto, R., Panichella, A., Panichella, S., 2013b. Improving IR-based traceability recovery via noun-based indexing of software artifacts. J. Softw. (Malden) 25 (7), 743–762.

Champagne, J.M., Carver, D.L., 2020. Discovering relationships among software artifacts. In: 2020 IEEE Aerospace Conference. IEEE, Big Sky, MT, USA, pp. 1–11. http://dx.doi.org/10.1109/AERO47225.2020.9172288.

Chen, X., Hosking, J., Grundy, J., Amor, R., 2018. DcTracVis: A system retrieving and visualizing traceability links between source code and documentation. Autom. Softw. Engg. 25 (4), 703–741.

Chen, L., Wang, D., Shi, L., Wang, Q., 2021. A self-enhanced automatic traceability link recovery via structure knowledge mining for small-scale labeled data. In: 2021 IEEE 45th Annual Computers, Software, and Applications Conference. COMPSAC, IEEE, Madrid, Spain, pp. 904–913. http://dx.doi.org/10.1109/COMPSAC51774.2021.00123.

Chen, L., Wang, D., Wang, J., Wang, Q., 2019. Enhancing unsupervised requirements traceability with sequential semantics. In: 2019 26th Asia-Pacific Software Engineering Conference. APSEC, IEEE, Putrajaya, Malaysia, pp. 23–30. http://dx.doi.org/10.1109/APSEC48747.2019.00013.

Cleland-Huang, J., Berenbach, B., Clark, S., Settimi, R., Romanova, E., 2007. Best practices for automated traceability. Computer 40 (6), 27–35.

Cleland-Huang, J., Chang, C., Christensen, M., 2003. Event-based traceability for managing evolutionary change. IEEE Trans. Softw. Eng. 29 (9), 796–810. http://dx.doi.org/10.1109/TSE.2003.1232285.

Cleland-Huang, J., Gotel, O.C.Z., Huffman Hayes, J., Mäder, P., Zisman, A., 2014. Software traceability: Trends and future directions. In: Future of Software Engineering Proceedings. In: FOSE 2014, Association for Computing Machinery, New York, NY, USA, pp. 55–69. http://dx.doi.org/10.1145/2593882.2593891.

Corriveau, J.-P., 1996. Traceability process for large OO projects. Computer 29 (9), 63–68. http://dx.doi.org/10.1109/2.536785.

Csuvik, V., Horváth, D., Horváth, F., Vidács, L., 2020. Utilizing source code embeddings to identify correct patches. In: 2020 IEEE 2nd International Workshop on Intelligent Bug Fixing. IBF, IEEE, London, ON, Canada, pp. 18–25. http://dx.doi.org/10.1109/IBF50092.2020.9034714.

Csuvik, V., Kicsi, A., Vidács, L., 2019a. Evaluation of textual similarity techniques in code level traceability. In: Misra, S., Gervasi, O., Murgante, B., Stankova, E., Korkhov, V., Torre, C., Rocha, A.M.A., Taniar, D., Apduhan, B.O., Tarantino, E. (Eds.), Computational Science and Its Applications — ICCSA 2019. Springer International Publishing, Cham, pp. 529–543.

Csuvik, V., Kicsi, A., Vidács, L., 2019b. Source code level word embeddings in aiding semantic test-to-code traceability. In: 2019 IEEE/ACM 10th International Symposium on Software and Systems Traceability. SST, IEEE, Montreal, QC, Canada, pp. 29–36. http://dx.doi.org/10.1109/SST.2019.00016.

Dasgupta, T., Grechanik, M., Moritz, E., Dit, B., Poshyvanyk, D., 2013. Enhancing software traceability by automatically expanding Corpora with relevant documentation. In: 2013 IEEE International Conference on Software Maintenance. IEEE, Eindhoven, Netherlands, pp. 320–329. http://dx.doi.org/10.1109/ICSM.2013.43.

Devlin, J., Chang, M., Lee, K., Toutanova, K., 2018. BERT: pre-training of deep bidirectional transformers for language understanding. CoRR abs/1810.04805, arXiv:1810.04805.

Dilshener, T., Wermelinger, M., Yu, Y., 2017. Locating bugs without looking back. Autom. Softw. Eng. 25 (3), 383–434. http://dx.doi.org/10.1007/s10515-017-0226-1.

Divya, K., Subha, R., Palaniswami, S., 2014. Similar words identification using naive and TF-IDF method. Int. J. Inform. Technol. Comput. Sci. 6, 42–47. http://dx.doi.org/10.5815/ijitcs.2014.11.06.

Doddapaneni, S., Ramesh, G., Kunchukuttan, A., Kumar, P., Khapra, M.M., 2021. A primer on pretrained multilingual language models. CoRR abs/2107.00676, arXiv:2107.00676.

Duan, C., Laurent, P., Cleland-Huang, J., Kwiatkowski, C., 2009. Towards automated requirements prioritization and triage. Requir. Eng. 14 (2), 73–89.

Eder, S., Femmer, H., Hauptmann, B., Junker, M., 2015. Configuring latent semantic indexing for requirements tracing. In: 2015 IEEE/ACM 2nd International Workshop on Requirements Engineering and Testing. IEEE, Florence, Italy, pp. 27–33. http://dx.doi.org/10.1109/RET.2015.13.

Effa Bella, E., Creff, S., Gervais, M.-P., Bendraou, R., 2019. ATLaS: A framework for traceability links recovery combining information retrieval and semi-supervised techniques. In: 2019 IEEE 23rd International Enterprise Distributed Object Computing Conference. EDOC, IEEE, Paris, France, pp. 161–170. http://dx.doi.org/10.1109/EDOC.2019.00028.

Effa Bella, E., Gervais, M.-P., Bendraou, R., Wouters, L., Koudri, A., 2018. Semi-supervised approach for recovering traceability links in complex systems. In: 2018 23rd International Conference on Engineering of Complex Computer Systems. ICECCS, IEEE, Melbourne, VIC, Australia, pp. 193–196. http://dx.doi.org/10.1109/ICECCS2018.2018.00030.

Efstathiou, V., Chatzilenas, C., Spinellis, D., 2018. Word embeddings for the software engineering domain. In: Proceedings of the 15th International Conference on Mining Software Repositories. MSR '18, Association for Computing Machinery, New York, NY, USA, pp. 38–41. http://dx.doi.org/10.1145/3196398.3196448.

Eggensperger, K., Hutter, F., Hoos, H., Leyton-Brown, K., 2015. Efficient benchmarking of hyperparameter optimizers via surrogates. In: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 29, no. 1. Association for the Advancement of Artificial Intelligence (AAAI), http://dx.doi.org/10.1609/aaai.v29i1.9375.

Falessi, D., Penta, M.D., Canfora, G., Cantone, G., 2016. Estimating the number of remaining links in traceability recovery. Empir. Softw. Eng. 22, 996–1027.

Ferreira, F., Silva, L.L., Valente, M.T., 2021. Software engineering meets deep learning: A mapping study. In: Proceedings of the 36th Annual ACM Symposium on Applied Computing. Association for Computing Machinery, New York, NY, USA, pp. 1542–1549. http://dx.doi.org/10.1145/3412841.3442029.

Florez, J.M., 2019. Automated fine-grained requirements-to-code traceability link recovery. In: Proceedings of the 41st International Conference on Software Engineering: Companion Proceedings. ICSE '19, IEEE Press, Montreal, Quebec, Canada, pp. 222–225. http://dx.doi.org/10.1109/ICSE-Companion.2019.00087.

Gadelha, G., Ramalho, F., Massoni, T., 2021. Traceability recovery between bug reports and test cases-a Mozilla firefox case study. Autom. Softw. Eng. 28, http://dx.doi.org/10.1007/s10515-021-00287-w.

Gharibi, R., Rasekh, A.H., Sadreddini, M.H., Fakhrahmad, S.M., 2018. Leveraging textual properties of bug reports to localize relevant source files. Inf. Process. Manage. 54 (6), 1058–1076. http://dx.doi.org/10.1016/j.ipm.2018.07.004.

Guo, J., Cheng, J., Cleland-Huang, J., 2017a. Semantically enhanced software traceability using deep learning techniques. In: 2017 IEEE/ACM 39th International Conference on Software Engineering. ICSE, IEEE, Buenos Aires, Argentina, pp. 3–14.

Guo, J., Gibiec, M., Cleland-Huang, J., 2017b. Tackling the term-mismatch problem in automated trace retrieval. Empir. Softw. Eng. 22 (3), 1103–1142.

Gupta, S., Gupta, S.K., 2019. Natural language processing in mining unstructured data from software repositories: A review. Sadhana 44 (12), 1–17, URL http://search.ebscohost.com.proxy-ub.rug.nl/login.aspx?direct=true&db=aph&AN=140970500&site=ehost-live&scope=site.

Hariri, R.H., Fredericks, E.M., 2018. Towards traceability link recovery for self-adaptive systems. In: Workshops At the Thirty-Second AAAI Conference on Artificial Intelligence. AAAI Press, Louisiana, USA.

Harzing, A.W.K., van der Wal, R., 2008. Google scholar as a new source for citation analysis. Ethics Sci. Environ. Polit. 8, 61–73.

Heck, P., Zaidman, A., 2014. Horizontal traceability for just-in-time requirements: The case for open source feature requests. J. Softw. Evol. Process 26 (12), 1280–1296. http://dx.doi.org/10.1002/smr.1678.

Hey, T., 2019. INDIRECT: Intent-driven requirements-to-code traceability. In: 2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings. ICSE-Companion, IEEE, Montreal, QC, Canada, pp. 190–191. http://dx.doi.org/10.1109/ICSE-Companion.2019.00078.

Hey, T., Chen, F., Weigelt, S., Tichy, W.F., 2021. Improving traceability link recovery using fine-grained requirements-to-code relations. In: 2021 IEEE International Conference on Software Maintenance and Evolution. ICSME, IEEE, Luxembourg, pp. 12–22. http://dx.doi.org/10.1109/ICSME52107.2021.00008.

Hirsch, T., Hofer, B., 2022. Detecting non-natural language artifacts for de-noising bug reports. Autom. Softw. Eng. 29 (2), http://dx.doi.org/10.1007/s10515-022-00350-0.

Hosseinzadeh Vahid, A., Arora, P., Liu, Q., Jones, G.J., 2015. A comparative study of online translation services for cross language information retrieval. In: Proceedings of the 24th International Conference on World Wide Web. In: WWW '15 Companion, Association for Computing Machinery, New York, NY, USA, pp. 859–864. http://dx.doi.org/10.1145/2740908.2743008.

Huang, Y., Liu, Z., Chen, X., Luo, X., 2016. Automatic matching release notes and source code by generating summary for software change. In: 2016 6th International Conference on Digital Home. ICDH, IEEE, Guangzhou, China, pp. 104–109. http://dx.doi.org/10.1109/ICDH.2016.031.

Iammarino, M., Aversano, L., Bernardi, M.L., Cimitile, M., 2020. A topic modeling approach to evaluate the comments consistency to source code. In: 2020 International Joint Conference on Neural Networks. IJCNN, IEEE, Glasgow, UK, pp. 1–8. http://dx.doi.org/10.1109/IJCNN48605.2020.9207651.

Javed, M.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. pp. 1–10.

Jiang, B., Liu, P., Xu, J., 2020. A deep learning approach to locate buggy files. In: 2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies. DESSERT, IEEE, Kyiv, Ukraine, pp. 219–223. http://dx.doi.org/10.1109/DESSERT50317.2020.9125003.

Kchaou., D., Bouassida., N., Ben-Abdallah., H., 2017. A new approach for traceability between UML models. In: Proceedings of the 12th International Conference on Software Technologies. ICSOFT, SciTePress, INSTICC, Madrid, Spain, pp. 128–139. http://dx.doi.org/10.5220/0006430001280139.

Kchaou, D., Bouassida, N., Mefteh, M., Ben-Abdallah, H., 2019. Recovering semantic traceability between requirements and design for change impact analysis. Innov. Syst. Softw. Eng. 15 (2), 101–115. http://dx.doi.org/10.1007/s11334-019-00330-w.

Keim, J., Kaplan, A., Koziolek, A., Mirakhorli, M., 2020a. Using BERT for the Detection of Architectural Tactics in Code. Technical Report 2, Karlsruher Institut für Technologie (KIT), http://dx.doi.org/10.5445/IR/1000121031.

Keim, J., Kaplan, A., Koziolek, A., Mirakhorli, M., 2020b. Does BERT understand code? — an exploratory study on the detection of architectural tactics in code. In: Software Architecture: 14th European Conference. ECSA 2020, L'Aquila, Italy, September 14–18, 2020, Proceedings, Springer-Verlag, Berlin, Heidelberg, pp. 220–228. http://dx.doi.org/10.1007/978-3-030-58923-3_15.

Keim, J., Koziolek, A., 2019. Towards consistency checking between software architecture and informal documentation. In: 2019 IEEE International Conference on Software Architecture Companion. ICSA-C, IEEE, Hamburg, Germany, pp. 250–253. http://dx.doi.org/10.1109/ICSA-C.2019.00052.

Keim, J., Schulz, S., Fuchß, D., Kocher, C., Speit, J., Koziolek, A., 2021. Trace link recovery for software architecture documentation. In: Biffl, S., Navarro, E., Löwe, W., Sirjani, M., Mirandola, R., Weyns, D. (Eds.), Software Architecture. Springer International Publishing, Cham, pp. 101–116.

Khatiwada, S., Tushev, M., Mahmoud, A., 2017. Just enough semantics an information theoretic approach for IR-based software bug localization. Inf. Softw. Technol. 93, http://dx.doi.org/10.1016/j.infsof.2017.08.012.

Khurana, D., Koli, A., Khatter, K., Singh, S., 2017. Natural language processing: State of the art, current trends and challenges. CoRR abs/1708.05148, arXiv:1708.05148.

Kicsi, A., Csuvik, V., Vidács, L., 2021. Large scale evaluation of natural language processing based test-to-code traceability approaches. IEEE Access 9, 79089–79104. http://dx.doi.org/10.1109/ACCESS.2021.3083923.

Kicsi, A., Tóth, L., Vidács, L., 2018. Exploring the benefits of utilizing conceptual information in test-to-code traceability. In: 2018 IEEE/ACM 6th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering. RAISE, IEEE, Gothenburg, Sweden, pp. 8–14.

Kitchenham, B.A., Charters, S., 2007. Guidelines for performing Systematic Literature Reviews in Software Engineering. Technical Report EBSE 2007-001, Keele University and Durham University Joint Report, Keele University, URL https://www.elsevier.com/__data/promis_misc/525444systematicreviewsguide.pdf.

Lam, A.N., Nguyen, A.T., Nguyen, H.A., Nguyen, T.N., 2015. Combining deep learning with information retrieval to localize buggy files for bug reports (N). In: 2015 30th IEEE/ACM International Conference on Automated Software Engineering. ASE, IEEE, Lincoln, NE, USA, pp. 476–481. http://dx.doi.org/10.1109/ASE.2015.73.

Lapeña, R., Pérez, F., Cetina, C., 2017. On the influence of models-to-natural-language transformation in traceability link recovery among requirements and conceptual models. In: Proceedings of the ER Forum 2017 and the ER 2017 Demo Track. CEUR-ws, Valencia, Spain, pp. 285–298, URL http://ceur-ws.org/Vol-1979/#paper-18.

Lapeña, R., Pérez, F., Cetina, C., Pastor, O., 2019. Improving traceability links recovery in process models through an ontological expansion of requirements. In: Giorgini, P., Weber, B. (Eds.), Advanced Information Systems Engineering — 31st International Conference. CAiSE 2019, Rome, Italy, June 3-7, 2019, Proceedings, In: Lecture Notes in Computer Science, vol.11483, Springer, Rome, Italy, pp. 261–275. http://dx.doi.org/10.1007/978-3-030-21290-2_17.

Lauriola, I., Lavelli, A., Aiolli, F., 2022. An introduction to deep learning in natural language processing: Models, techniques, and tools. Neurocomputing 470, 443–456. http://dx.doi.org/10.1016/j.neucom.2021.05.103.

Li, Y., Cleland-Huang, J., 2013. Ontology-based trace retrieval. In: 2013 7th International Workshop on Traceability in Emerging Forms of Software Engineering. TEFSE, IEEE, San Francisco, CA, pp. 30–36. http://dx.doi.org/10.1109/TEFSE.2013.6620151.

Lin, J., Liu, Y., Guo, J., Cleland-Huang, J., Goss, W., Liu, W., Lohar, S., Monaikul, N., Rasin, A., 2017. Tiqi: A natural language interface for querying software project data. In: 2017 32nd IEEE/ACM International Conference on Automated Software Engineering. ASE, IEEE, Urbana, IL, USA, pp. 973–977. http://dx.doi.org/10.1109/ASE.2017.8115714.

Lin, J., Liu, Y., Zeng, Q., Jiang, M., Cleland-Huang, J., 2021. Traceability transformed: Generating more accurate links with pre-trained BERT models. In: 2021 IEEE/ACM 43rd International Conference on Software Engineering. ICSE, IEEE, Madrid, ES, pp. 324–335. http://dx.doi.org/10.1109/ICSE43902.2021.00040.

Lindvall, M., Feldmann, R.L., Karabatis, G., Chen, Z., Janeja, V.P., 2009. Searching for relevant software change artifacts using semantic networks. In: Proceedings of the 2009 ACM Symposium on Applied Computing. SAC '09, Association for Computing Machinery, New York, NY, USA, pp. 496–500. http://dx.doi.org/10.1145/1529282.1529387.

Liu, Y., Lin, J., Cleland-Huang, J., 2020a. Traceability support for multi-lingual software projects. CoRR abs/2006.16940, arXiv:2006.16940.

Liu, Y., Lin, J., Zeng, Q., Jiang, M., Cleland-Huang, J., 2020b. Towards semantically guided traceability. In: 2020 IEEE 28th International Requirements Engineering Conference. RE, IEEE, Zurich, Switzerland, pp. 328–333. http://dx.doi.org/10.1109/RE48521.2020.00043.

Liu, G., Lu, Y., Shi, K., Chang, J., Wei, X., 2019. Mapping bug reports to relevant source code files based on the vector space model and word embedding. IEEE Access 7, 78870–78881. http://dx.doi.org/10.1109/ACCESS.2019.2922686.

Mäder, P., Olivetto, R., Marcus, A., 2017. Empirical studies in software and systems traceability. Empirical Softw. Engg. 22 (3), 963–966. http://dx.doi.org/10.1007/s10664-017-9509-1.

Mahmood, K., Takahashi, H., Alobaidi, M., 2015. A semantic approach for traceability link recovery in aerospace requirements management system. In: 2015 IEEE Twelfth International Symposium on Autonomous Decentralized Systems. IEEE, Taichung, Taiwan, pp. 217–222. http://dx.doi.org/10.1109/ISADS.2015.33.

Mahmoud, A., 2015. An information theoretic approach for extracting and tracing non-functional requirements. In: 2015 IEEE 23rd International Requirements Engineering Conference. RE, IEEE, Ottawa, ON, Canada, pp. 36–45. http://dx.doi.org/10.1109/RE.2015.7320406.

Mahmoud, A., Bradshaw, G., 2015. Estimating semantic relatedness in source code. ACM Trans. Softw. Eng. Methodol. 25 (1), http://dx.doi.org/10.1145/2824251.

Mahmoud, A., Niu, N., 2015. On the role of semantics in automated requirements tracing. Requir. Eng. 20 (3), 281–300. http://dx.doi.org/10.1007/s00766-013-0199-y.

Mahmoud, A., Williams, G., 2016. Detecting, classifying, and tracing non-functional software requirements. Requir. Eng. 21 (3), 357–381. http://dx.doi.org/10.1007/s00766-016-0252-8.

Maletic, J.I., Munson, E.V., Marcus, A., Nguyen, T.N., 2003. Using a hypertext model for traceability link conformance analysis. In: In Proc. of the 2nd Int. Workshop on Traceability in Emerging Forms of Software Engineering. ACM, USA, pp. 47–54.

Malhotra, R., Aggarwal, S., Girdhar, R., Chugh, R., 2018. Bug localization in software using NSGA-II. In: 2018 IEEE Symposium on Computer Applications Industrial Electronics. ISCAIE, IEEE, Penang, Malaysia, pp. 428–433. http://dx.doi.org/10.1109/ISCAIE.2018.8405511.

Malik, K., Alobaidi, M., Takahashi, H., 2016. Autonomous decentralized semantic based traceability link recovery framework. IEICE Trans. Inf. Syst. E99.D, 2283–2294. http://dx.doi.org/10.1587/transinf.2016EDP7018.

Mäntylä, M.V., Calefato, F., Claes, M., 2018. Natural language or not (NLON): A package for software engineering text analysis pipeline. In: Proceedings of the 15th International Conference on Mining Software Repositories. MSR '18, Association for Computing Machinery, New York, NY, USA, pp. 387–391. http://dx.doi.org/10.1145/3196398.3196444.

Marcus, A., Maletic, J.I., 2003. Recovering documentation-to-source-code traceability links using latent semantic indexing. In: 25th International Conference on Software Engineering. 2003. Proceedings., IEEE, Portland, OR, USA, pp. 125–135.

Mills, C., Haiduc, S., 2017. The impact of retrieval direction on IR-based traceability link recovery. In: 2017 IEEE/ACM 39th International Conference on Software Engineering: New Ideas and Emerging Technologies Results Track. ICSE-NIER, IEEE, Buenos Aires, Argentina, pp. 51–54. http://dx.doi.org/10.1109/ICSE-NIER.2017.14.

Mishra, A., Dubey, D., 2013. A comparative study of different software development life cycle models in different scenarios. Int. J. Adv. Res. Comput. Sci. Manag. Stud. 1 (5), 64–69.

Moreno, L., Bandara, W., Haiduc, S., Marcus, A., 2013. On the relationship between the vocabulary of bug reports and source code. In: 2013 IEEE International Conference on Software Maintenance. pp. 452–455. http://dx.doi.org/10.1109/ICSM.2013.70.

Moreno Lopez, M., Kalita, J., 2017. Deep Learning applied to NLP. ArXiv E-Prints, arXiv:1703.03091.

Mustafa, N., Labiche, Y., 2017. The need for traceability in heterogeneous systems: A systematic literature review. In: 2017 IEEE 41st Annual Computer Software and Applications Conferencen, Vol. 1. COMPSAC, IEEE, Turin, Italy, pp. 305–310. http://dx.doi.org/10.1109/COMPSAC.2017.237.

Nadkarni, P.M., Ohno-Machado, L., Chapman, W.W., 2011. Natural language processing: An introduction. J. Am. Med. Inform. Assoc. 18 (5), 544–551. http://dx.doi.org/10.1136/amiajnl-2011-000464.

Navarro-Almanza, R., Juarez-Ramirez, R., Licea, G., 2017. Towards supporting software engineering using deep learning: A case of software requirements classification. In: 2017 5th International Conference in Software Engineering Research and Innovation. CONISOFT, IEEE, Merida, Mexico, pp. 116–120. http://dx.doi.org/10.1109/CONISOFT.2017.00021.

Nishikawa, K., Washizaki, H., Fukazawa, Y., Oshima, K., Mibe, R., 2015. Recovering transitive traceability links among software artifacts. In: 2015 IEEE International Conference on Software Maintenance and Evolution. ICSME, IEEE, Bremen, Germany, pp. 576–580. http://dx.doi.org/10.1109/ICSM.2015.7332517.

Panichella, A., De Lucia, A., Zaidman, A., 2015. Adaptive user feedback for IR-based traceability recovery. In: 2015 IEEE/ACM 8th International Symposium on Software and Systems Traceability. IEEE, Florence, Italy, pp. 15–21. http://dx.doi.org/10.1109/SST.2015.10.

Panichella, A., Dit, B., Oliveto, R., Di Penta, M., Poshynanyk, D., De Lucia, A., 2013. How to effectively use topic models for software engineering tasks? An approach based on genetic algorithms. In: 2013 35th International Conference on Software Engineering. ICSE, IEEE, San Francisco, CA, pp. 522–531. http://dx.doi.org/10.1109/ICSE.2013.6606598.

Pauzi, Z., Capiluppi, A., 2020. Text similarity between concepts extracted from source code and documentation. In: Intelligent Data Engineering and Automated Learning — IDEAL 2020: 21st International Conference. Springer-Verlag, Berlin, Heidelberg, pp. 124–135. http://dx.doi.org/10.1007/978-3-030-62362-3_12.

Pauzi, Z., Capiluppi, A., 2021. Extracting and comparing concepts emerging from software code, documentation and tests. In: Catolino , G., Di Nucci , D., Tamburri , D. (Eds.), 20th Belgium-Netherlands Software Evolution Workshop, BENEVOL 2021V. In: CEUR Workshop Proceedings, CEUR Workshop Proceedings, 's-Hertogenbosch, Netherlands, pp. 1–11.

Pete, I., Balasubramaniam, D., 2015. Handling the differential evolution of software artefacts: A framework for consistency management. In: 2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering. SANER, IEEE, Montreal, QC, Canada, pp. 599–600. http://dx.doi.org/10.1109/SANER.2015.7081889.

Petersen, K., Vakkalanka, S., Kuzniarz, L., 2015. Guidelines for conducting systematic mapping studies in software engineering: An update. Inf. Softw. Technol. 64, 1–18. http://dx.doi.org/10.1016/j.infsof.2015.03.007.

Poshyvanyk, D., Gethers, M., Marcus, A., 2013. Concept location using formal concept analysis and information retrieval. ACM Trans. Softw. Eng. Methodol. 21 (4), http://dx.doi.org/10.1145/2377656.2377660.

Pruski, P., Lohar, S., Aquanette, R., Ott, G., Amornborvornwong, S., Rasin, A., Cleland-Huang, J., 2014. Tiqi: Towards natural language trace queries. In: 2014 IEEE 22nd International Requirements Engineering Conference. RE, IEEE, Karlskrona, Sweden, pp. 123–132. http://dx.doi.org/10.1109/RE.2014.6912254.

Pruski, P., Lohar, S., Goss, W., Rasin, A., Cleland-Huang, J., 2015. Tiqi: answering unstructured natural language trace queries. Requir. Eng. 20, 215–232.

Qusef, A., Bavota, G., Oliveto, R., De Lucia, A., Binkley, D., 2014. Recovering test-to-code traceability using slicing and textual analysis. J. Syst. Softw. 88 (C), 147–168.

Rasekh, A.H., Arshia, A.H., Fakhrahmad, S.M., Sadreddini, M.H., 2017. Mining and discovery of hidden relationships between software source codes and related textual documents. Digit. Scholarsh. Humanit. 33 (3), 651–669. http://dx.doi.org/10.1093/llc/fqx052.

Rasekh, A., Fakhrahmad, S., Sadreddini, M., 2019. Mining traces between source code and textual documents. Int. J. Comput. Appl. Technol. 59 (1), 43–52. http://dx.doi.org/10.1504/IJCAT.2019.097116.

Regan, G., Mc Caffery, F., Mc Daid, K., Flood, D., 2013. Medical device standards' requirements for traceability during the software development lifecycle and implementation of a traceability assessment model. Comput. Stand. Interfaces 36 (1), 3–9. http://dx.doi.org/10.1016/j.csi.2013.07.012.

Rubasinghe, I., Meedeniya, D., Perera, I., 2018a. Automated inter-artefact traceability establishment for DevOps practice. In: 2018 IEEE/ACIS 17th International Conference on Computer and Information Science. ICIS, IEEE, Singapore, pp. 211–216. http://dx.doi.org/10.1109/ICIS.2018.8466414.

Rubasinghe, I.D., Meedeniya, D.A., Perera, I., 2018b. Software artefact traceability analyser: A case-study on POS system. In: Proceedings of the 6th International Conference on Communications and Broadband Networking. In: ICCBN 2018, Association for Computing Machinery, New York, NY, USA, pp. 1–5. http://dx.doi.org/10.1145/3193092.3193094.

Rubasinghe, I., Meedeniya, D., Perera, I., 2020. Tool Support for Software Artefact Traceability in DevOps Practice: SAT-Analyser. IGI Global, USA, pp. 130–167. http://dx.doi.org/10.4018/978-1-7998-1863-2.ch005.

Russell-Rose, T., Stevenson, M., 2009. The role of natural language processing in information retrieval: Searching for meaning and structure. In: Information Retrieval: Searching in the 21st Century. Wiley Telecom, UK, pp. 215–231. http://dx.doi.org/10.1002/9780470033647.ch10.

Saha, R.K., Lease, M., Khurshid, S., Perry, D.E., 2013. Improving bug localization using structured information retrieval. In: 2013 28th IEEE/ACM International Conference on Automated Software Engineering. ASE, pp. 345–355. http://dx.doi.org/10.1109/ASE.2013.6693093.

Salih, O., Sahraoui, A.-E.-K., 2018. Toward requirements and design traceability using natural language processing. Eur. J. Eng. Res. Sci. 3, http://dx.doi.org/10.24018/ejers.2018.3.7.807.

Salih, O., Sahraoui, A.-E.-K., Mahmoud, M., Babiker, A.-E.-A., 2021. Requirements and design consistency: A Bi-directional traceability and natural language processing assisted approach. Eur. J. Eng. Technol. Res. 6, 55–64. http://dx.doi.org/10.24018/ejers.2021.6.3.2373.

Sawant, A.A., Devanbu, P., 2021. Naturally!: How breakthroughs in natural language processing can dramatically help developers. IEEE Softw. 38 (5), 118–123. http://dx.doi.org/10.1109/MS.2021.3086338.

Scanniello, G., Marcus, A., Pascale, D., 2015. Link analysis algorithms for static concept location: An empirical assessment. Empir. Softw. Eng. 20, http://dx.doi.org/10.1007/s10664-014-9327-7.

Schwarz, H., Ebert, J., Winter, A., 2010. Graph-based traceability: A comprehensive approach. Softw. Syst. Model. 9 (4), 473–492.

Shokripour, R., Anvik, J., Kasirun, Z.M., Zamani, S., 2013. Why so complicated? Simple term filtering and weighting for location-based bug report assignment recommendation. In: 2013 10th Working Conference on Mining Software Repositories. MSR, IEEE, San Francisco, CA, pp. 2–11. http://dx.doi.org/10.1109/MSR.2013.6623997.

Singh, M., 2022. Using natural language processing and graph mining to explore inter- related requirements in software artefacts. SIGSOFT Softw. Eng. Notes 44 (1), 37–42. http://dx.doi.org/10.1145/3310013.3310018.

Snow, R., O'connor, B., Jurafsky, D., Ng, A.Y., 2008. Cheap and fast–but is it good? evaluating non-expert annotations for natural language tasks. In: Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing. pp. 254–263.

Spanoudakis, G., Garcez, A., Zisman, A., 2003. Revising rules to capture requirements traceability relations: A machine learning approach. In: Proceedings of the Fifteenth International Conference on Software Engineering & Knowledge Engineering. SEKE'2003, Knowledge Systems Institute, San Francisco Bay, CA, USA, pp. 570–577.

Tantithamthavorn, C.K., Jiarpakdee, J., 2021. Explainable AI for software engineering. In: 2021 36th IEEE/ACM International Conference on Automated Software Engineering. ASE, pp. 1–2. http://dx.doi.org/10.1109/ASE51524.2021.9678580.

Thommazo, A.D., Ribeiro, T., Olivatto, G., Werneck, V., Fabbri, S., 2013. An automatic approach to detect traceability links using fuzzy logic. In: 2013 27th Brazilian Symposium on Software Engineering. IEEE, Brasilia, Brazil, pp. 21–30. http://dx.doi.org/10.1109/SBES.2013.11.

Thommazo, A.D., Rovina, R., de Oliveira, T.R., Olivatto, G., Hernandes, E.M., Werneck, V., Fabbri, S.C.P.F., 2014. Using artificial intelligence techniques to enhance traceability links. In: ICEIS, Vol. 1. SCITEPRESS — Science and Technology Publications, Lisbon, Portugal, pp. 26–38.

Tian, Q., Cao, Q., Sun, Q., 2018. Adapting word embeddings to traceability recovery. In: 2018 International Conference on Information Systems and Computer Aided Education. ICISCAE, IEEE, Changchun, China, pp. 255–261. http://dx.doi.org/10.1109/ICISCAE.2018.8666883.

Tichy, W.F., Landhäußer, M., Körner, S.J., 2013. Universal Programmability — How AI Can Help. Technical Report 15, Karlsruher Institut für Technologie (KIT), http://dx.doi.org/10.5445/IR/1000037684.

Torfi, A., Shirvani, R.A., Keneshloo, Y., Tavaf, N., Fox, E.A., 2021. Natural language processing advancements by deep learning: A survey. arXiv:2003.01200.

Tsuchiya, R., Washizaki, H., Fukazawa, Y., Oshima, K., Mibe, R., 2015. Interactive recovery of requirements traceability links using user feedback and configuration management logs. In: Zdravkovic, J., Kirikova, M., Johannesson, P. (Eds.), Advanced Information Systems Engineering. Springer International Publishing, Cham, pp. 247–262.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I., 2017. Attention is all you need. CoRR abs/1706.03762, arXiv:1706.03762.

Velasco, A., Aponte Melo, J.H., 2019. Recovering fine grained traceability links between software mandatory constraints and source code. In: Florez, H., Leon, M., Diaz-Nafria, J.M., Belli, S. (Eds.), Applied Informatics. Springer International Publishing, Cham, pp. 517–532.

Vrettas, G., Sanderson, M., 2015. Conferences versus journals in computer science: Conferences vs. journals in computer science. J. Assoc. Inform. Sci. Technol. 66, http://dx.doi.org/10.1002/asi.23349.

Wang, X., Cao, Q., Sun, Q., 2019. An improved approach based on balanced keyword weight to traceability recovery. IOP Conf. Ser. Mater. Sci. Eng. 569, 052109. http://dx.doi.org/10.1088/1757-899X/569/5/052109.

Wang, S., Lo, D., Lawall, J., 2014. Compositional vector space models for improved bug localization. In: 2014 IEEE International Conference on Software Maintenance and Evolution. IEEE, Victoria, BC, Canada, pp. 171–180. http://dx.doi.org/10.1109/ICSME.2014.39.

Wang, W., Niu, N., Liu, H., Niu, Z., 2018. Enhancing automated requirements traceability by resolving polysemy. In: 2018 IEEE 26th International Requirements Engineering Conference. RE, IEEE, Banff, AB, Canada, pp. 40–51. http://dx.doi.org/10.1109/RE.2018.00-53.

Wijesinghe, D.B., Kamalabalan, K., Uruththirakodeeswaran, T., Thiyagalingam, G., Perera, I., Meedeniya, D., 2014. Establishing traceability links among software artefacts. In: 2014 14th International Conference on Advances in ICT for Emerging Regions. ICTer, IEEE, Colombo, Sri Lanka, pp. 55–62. http://dx.doi.org/10.1109/ICTER.2014.7083879.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Scao, T.L., Gugger, S., Drame, M., Lhoest, Q., Rush, A.M., 2019. HuggingFace's transformers: State-of-the-art natural language processing. http://dx.doi.org/10.48550/ARXIV.1910.03771, arXiv.

Xia, X., Lo, D., Wang, X., Zhang, C., Wang, X., 2014. Cross-language bug localization. In: Proceedings of the 22nd International Conference on Program Comprehension. In: ICPC 2014, Association for Computing Machinery, New York, NY, USA, pp. 275–278. http://dx.doi.org/10.1145/2597008.2597788.

Xie, R., Chen, L., Ye, W., Li, Z., Hu, T., Du, D., Zhang, S., 2019. DeepLink: A code knowledge graph based deep learning approach for issue-commit link recovery. In: 2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering. SANER, IEEE, Hangzhou, China, pp. 434–444. http://dx.doi.org/10.1109/SANER.2019.8667969.

Yalla, P., Sharma, N., 2015. Integrating natural language processing and software engineering. Int. J. Softw. Eng. Appl. 9, 127–136. http://dx.doi.org/10.14257/ijseia.2015.9.11.12.

Yang, G., Lee, B., 2021. Utilizing topic-based similar commit information and CNN-LSTM algorithm for bug localization. Symmetry 13 (3), 406. http://dx.doi.org/10.3390/sym13030406.

Yıldız, O.T., Okutan, A., Solak, E., 2014. Bilingual software requirements tracing using vector space model.

Zamani, S., Lee, S.P., Shokripour, R., Anvik, J., 2014. A noun-based approach to feature location using time-aware term-weighting. Inf. Softw. Technol. 56 (8), 991–1011. http://dx.doi.org/10.1016/j.infsof.2014.03.007.

Zhang, Y., Lo, D., Xia, X., Le, T.B., Scanniello, G., Sun, J., 2016a. Inferring links between concerns and methods with multi-abstraction vector space model. In: 2016 IEEE International Conference on Software Maintenance and Evolution. ICSME 2016, Raleigh, NC, USA, October 2-7, 2016, IEEE Computer Society, Raleigh, NC, USA, pp. 110–121. http://dx.doi.org/10.1109/ICSME.2016.51.

Zhang, J.-X., Tao, C.-Q., Huang, Z.-Q., Chen, X., 2021. Discovering API directives from API specifications with text classification. J. Comput. Sci. Tech. 36 (4), 922–943.

Zhang, Y., Wan, C., Jin, B., 2016b. An empirical study on recovering requirement-to-code links. In: 2016 17th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing. SNPD, IEEE, Shanghai, China, pp. 121–126. http://dx.doi.org/10.1109/SNPD.2016.7515889.

Zhao, T., Cao, Q., Sun, Q., 2017a. An improved approach to traceability recovery based on word embeddings. In: 2017 24th Asia-Pacific Software Engineering Conference. APSEC, IEEE, Nanjing, China, pp. 81–89.

Zhao, T., Cao, Q., Sun, Q., 2017b. An improved approach to traceability recovery based on word embeddings. In: 2017 24th Asia-Pacific Software Engineering Conference. APSEC, IEEE, Nanjing, China, pp. 81–89. http://dx.doi.org/10.1109/APSEC.2017.14.

Zhou, X., Jin, Y., Zhang, H., Li, S., Huang, X., 2016. A map of threats to validity of systematic literature reviews in software engineering. In: 2016 23rd Asia-Pacific Software Engineering Conference. APSEC, IEEE, Hamilton, New Zealand, pp. 153–160. http://dx.doi.org/10.1109/APSEC.2016.031.

Zhou, Y., Yanxiang, T., Chen, T., Han, J., 2017. Augmenting bug localization with part-of-speech and invocation. Int. J. Softw. Eng. Knowl. Eng. 27, 925–949. http://dx.doi.org/10.1142/S0218194017500346.

Zaki Pauzi is a data scientist at British Petroleum plc (bp), specialised in the field of natural language processing (NLP). His work involves developing data-driven solutions based on natural language data within the Energy sector. He has completed his bachelor's degree (with honours) in Computer Science with Management at King's College London back in 2019.

Zaki is also now in his third year of Ph.D. studies at University of Groningen, supervised by Prof. Dr. Andrea Capiluppi. His doctorate research is in the field of NLP in software engineering, with a focus on software traceability. Previous publications include "Text Similarity between Concepts Extracted from Source Code and Documentation" (IDEAL 2020) and "Extracting and Comparing Concepts Emerging from Software Code, Documentation and Tests" (CEUR workshop proceedings).

Linkedin: https://www.linkedin.com/in/zaki-pauzi-363a90128