Linking User Stories and Source Code Commits

Vasilka Saklamaeva[1,*], Luka Pavlič[1]

[1]University of Maribor, Faculty of Electrical Engineering and Computer Science, Koroška cesta 46, 2000 Maribor, Slovenia;

Abstract

User stories are widely used in agile software development to capture requirements due to their simplicity and adaptability. However, as development progresses, the relationship between user stories and the source code implementing them often becomes unclear, creating a traceability challenge. This paper presents a systematic literature review of traceability between user stories and source code commits in agile projects. A total of 51 studies were analyzed to identify approaches and techniques for establishing these links. The findings show that traceability can be achieved through direct methods based on semantic similarity between requirements and code artifacts, as well as indirect methods that leverage intermediate artifacts such as issue trackers, architectural models, and commit metadata. The reviewed techniques have evolved from traditional Information Retrieval models, including Vector Space Models and Latent Semantic Indexing, to Machine Learning approaches and, more recently, Deep Learning and Large Language Models. These advances demonstrate a growing trend toward more accurate, automated, and scalable traceability solutions in modern software engineering.

Keywords

software traceability, requirements traceability, traceability recovery, agile software development, requirements engineering

1. Introduction

User stories have become the de facto standard for capturing requirements in agile software development. A user story is a semi-structured natural language specification that follows a template such as *"As a [type of user], I want [goal], so that [benefit/reason]"*, making requirements accessible to both technical and non-technical stakeholders [1]. Because they are lightweight and easy to modify, user stories enable the iterative, feedback-driven cycles that define agile methodologies. However, a user story is not a complete requirements specification; it is deliberately lightweight, like a trigger for conversation.

As implementation progresses, the requirements captured in that original trigger become scattered across multiple commits, each addressing part of the story or refining it through bug fixes. The connection between what was agreed upon and what was actually built grows increasingly unclear. This is the problem of **software traceability**: the maintenance of relationships between development artifacts, especially the link between requirements and their implementation [2]. Traceability links between user stories and source code commits support important development activities, including change impact analysis, program comprehension, progress tracking, and compliance validation [3].

Despite its importance, traceability between user stories and source code is rarely practiced consistently. Studies have found that only 42-60% of commits contain an issue or user story identifier in their commit message [4, 2], and over half of pull requests are submitted without any link to an issue [2]. The primary issue of establishing and maintaining trace links manually is time-consuming and prone to errors, while automated tools remain rare and are not yet trusted to replace human judgment [5]. The gap between the need for traceability and its inadequate translation in practice has motivated a substantial amount of research over the past two decades.

This paper does not address user story mining in the strict sense, which is the automated extraction or reconstruction of user stories from project artifacts. A prerequisite for that task is the existing reliable traceability between stories and commits, which is precisely what is missing in most projects. Instead,

SQAMIA 2026: Workshop on Software Quality, Analysis, Monitoring, Improvement, and Applications, September 16–18, 2026, Niš, Serbia

*Corresponding author.

vasilka.saklamaeva@um.si (V. Saklamaeva); luka.pavlic@um.si (L. Pavlič)

0009-0005-0221-4840 (V. Saklamaeva); 0000-0001-5477-4747 (L. Pavlič)



© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073

published 2026-11-19

CEUR-WS.org/Vol-4336/paper17.pdf

this review examines the foundational layer: the techniques, information sources, and tools that can establish and maintain traceability links between user stories and commits. Given the prevalence of agile development, a significant portion of the discussed research focuses on the **Git ecosystem**, giving access to rich metadata and natural language found in issues, pull requests, and commit messages to bridge the gap between requirements and implementation. Such links are a prerequisite for any meaningful user story mining effort, as they provide the mapped data from which patterns, completions, and story-to-code relationships can be derived.

This paper presents a systematic literature review that examines the problem of traceability between user stories and source code commits in agile software projects. The review is guided by two research questions:

> **RQ1: How can user stories be linked to source code commits?** This question explores the information sources and approaches available for establishing traceability links, such as commit messages, issue tracker references and logs. It also asks what types of information are most effective for creating reliable links.
>
> **RQ2: What techniques exist for establishing traceability between user stories and source code commits?** This question explores the technical aspects, covering machine learning, natural language processing, information retrieval, and fuzzy matching approaches. We are also interested to examine the tools that implement these techniques, with a comparative analysis of their functionalities and limitations.

By reviewing and categorizing the state of the art, this paper aims to provide researchers and practitioners with an understanding of the available methods, the evidence for their effectiveness, and their limitations.

The rest of the paper is structured as follows: We start with an overview of related works in this area, which are presented in Section 2. In Section 3 we present the research process, including the literature selection criteria and requirements, as well as the final set of papers that our findings are be based on. Section 4 addresses the first research question. It comprises Subsections 4.1, 4.2, and 4.3, which examine information sources, approaches for establishing traceability links, and the most effective types of information for traceability, respectively. In the following Section 5, we present the findings for our second research question. Here we focus on the technical approaches (Subsection 5.1), the representative tools (Subsection 5.2) and their limitations for their implementation (Subsection 5.3). Our overall findings and a condensed presentation of the answers to the set research questions are presented in Section 6. We conclude our paper and provide ideas for further research in Section 7.

2. Related work

Several secondary studies have studied the broader traceability landscape. Chen et al. constructed the first global artifacts traceability graph across 76 studies, finding that the requirements-to-code pair is the most intensively studied but that approximately 45% of artifact pairs appear only once [6]. Aung et al. reviewed 33 studies on automatic traceability link recovery for change impact analysis, which concluded with a statement that there is a scarcity of industrial datasets [7].

Two SLRs directly address user stories as requirements artifacts. Raharjana et al. covered 38 studies on NLP for user stories, identifying four application categories including trace link creation [1]. Lyu et al. reviewed 40 studies on issue-based requirement traceability, finding that issues are linked to commits, code, reviews, and test cases via ML and IR techniques [8].

More recent reviews have addressed the appearance of NLP and LLM in this domain. Khalil et al. mapped 49 studies on NLP for requirements traceability, categorizing approaches into ontology-based and embedding techniques [9]. Sonbol et al. mapped 104 papers on NLP text representations for RE tasks, showing that traceability is among the most frequently addressed applications [10]. Rizvi mapped 132 studies on LLMs for traceability link discovery, finding that 44% use prompting, 29% fine-tuning, and 13% RAG, with requirements-to-code as the most common direction [11].

While several secondary studies have mapped the wider traceability landscape, this work differentiates itself by focusing on the specific intersection of user story templates and Git-centric metadata. Existing reviews such as those by Raharjana et al. [1] address user stories primarily through the lens of NLP applications (e.g., quality assessment) rather than technical link recovery, while Lyu et al. [8] collect all issue types, overlooking the unique linguistic constraints (e.g., As-a/I-want) and the role of PRs as intermediate artifacts in the Git ecosystem. Moreover, this paper synthesizes the most recent high-performance frameworks (specifically Retrieval-Augmented Generation models like LiSSA and knowledge distillation approaches like MTLink), which represent a significant step beyond the traditional embedding and prompting techniques covered in earlier surveys by Khalil et al. [9] and Rizvi et al. [11]. By positioning this traceability as the first layer for user story mining, this review provides a technical roadmap for reconstructing agile requirements from large-scale open-source repositories.

User stories and bug reports serve different purposes in agile development. User stories describe new functionality or user requirements to be implemented, whereas bug reports document defects that require correction after implementation. As this review focuses on traceability between agile work items and source code commits, the primary focus was on user stories and related planning artifacts (requirements, issues, features...). Studies addressing only bug report–commit traceability were not included unless they contributed to the broader context of agile work-item traceability.

3. Research method

In order to answer both of our research questions presented in Section 1, we conducted a Systematic Literature Review (SLR) in the same manner as proposed by Kitchenham et al. [12]. The literature search process was conducted according to the following criteria and requirements:

- We limited ourselves to the results found in five academic digital libraries: IEEE Xplore, ACM, Springer, Science Direct and Web of Science.
- The keywords used for the literature search for the appropriate research questions were:
    1. **RQ1:** (("All Metadata":„user stor*") OR ("All Metadata":„requirement*") OR ("All Metadata":„issue*") OR ("All Metadata":„featur*") OR ("All Metadata":„commit")) AND (("All Metadata":„agile") OR ("All Metadata":„code") OR ("All Metadata":„software")) AND (("All Metadata":„trace*") OR ("All Metadata":„extract") OR ("All Metadata":„link") OR ("All Metadata":„retriev*") OR ("All Metadata":„map") OR ("All Metadata":„min*"))
    2. **RQ2:** (("All Metadata":„user stor*") OR ("All Metadata":„requirement*") OR ("All Metadata":„issue*") OR ("All Metadata":„featur*") OR ("All Metadata":„commit")) AND (("All Metadata":„agile") OR ("All Metadata":„code") OR ("All Metadata":„software")) AND (("All Metadata":„trace*") OR ("All Metadata":„extract") OR ("All Metadata":„link") OR ("All Metadata":„retriev*") OR ("All Metadata":„map") OR ("All Metadata":„min*")) AND (("All Metadata":„tool") OR ("All Metadata":„technique") OR ("All Metadata":„method") OR ("All Metadata":„approach"))
- The literature reviewed was limited to the last five years (2021 to 2026).
- The obtained results were limited to peer-reviewed literature.
- All relevant literature needed to be written in English.
- Retrieved literature that was not directly connected to answering the presented research questions was excluded.

The search strings intentionally include broader terms such as "requirements", "issues", and "features", since different papers use different terminology to describe similar concepts. However, during the screening process, only papers that focused on traceability between user stories (or similar **agile** work items) and source code commits were included, based on the predefined criteria and requirements. The literature review was conducted in mid-2026 and consisted of six stages, which are presented below.

1. **Initial search.**
   In the first phase of the SLR process we obtained **895** items. In the digital libraries ScienceDirect, SpringerLink and ACM Digital Library we selected papers in the fields of Computer Science, Informatics and Software Engineering, and we filtered the search strings according to the set inclusion criteria. In Table 1 we present the number of records initially retrieved from each of the five chosen digital libraries.

Table 1
Search results by digital library and research question

Digital Library	Hits RQ1	Hits RQ2	Sum
ACM Digital Library	114	120	234
IEEE Xplore	34	54	88
SpringerLink	56	75	131
ScienceDirect	29	31	60
Web of Science	381	1	382
Total			895

2. **Duplicate removal.**
   The process of duplicate removal followed, where all results were normalised in a single literature list and duplicates were eliminated. We identified 203 instances where papers showed up multiple times, and, after their removal, the literature list was reduced to **692** items.
3. **Title and keyword-based screening.**
   In this stage we eliminated the largest amount of papers - out of 692 items we reduced the literature list to **29** items.
4. **Abstract-based screening.**
   The wider range of papers was analysed additionally, according to the abstract of the paper. Our literature list was reduced from 29 to **21** items.
5. **Content-based screening.**
   In this stage we carried out a thorough review of the selected papers based on their content. We analysed 21 papers and none were eliminated.
6. **Snowballing.**
   In addition, we reviewed references of the selected papers, as well as papers referencing the set of literature, to identify additional sources. This way, we were certain that we also included sources that were not found in the selected databases, but met the set criteria and addressed our research area. At this stage we used ResearchRabbit, which is a tool that presents connected papers in a collection. The publication year (2021-2026) and peer-review restrictions applied only to the initial database search. Additional studies identified through **backward and forward snowballing** were assessed based on their relevance to the research questions and were included if they provided important contributions. After a review of the potential candidates, we added a total of 30 items, therefore bringing our final set of literature to **51** primary sources.

Figure 1 presents the SLR process and the literature identified during each phase. During the title- and keyword-based screening, studies were excluded only when it was clear from the title and keywords that they were unrelated to the research topic. In cases of uncertainty, papers were retained for the abstract screening phase to minimize the risk of excluding relevant studies. The literature search addressing both research questions resulted in the process shown in Figure 2, which summarizes the identified information sources, technical approaches, and tools. Following the selection process described in Section 3, we collected the final set of **51** primary studies, which are listed in Table 2.

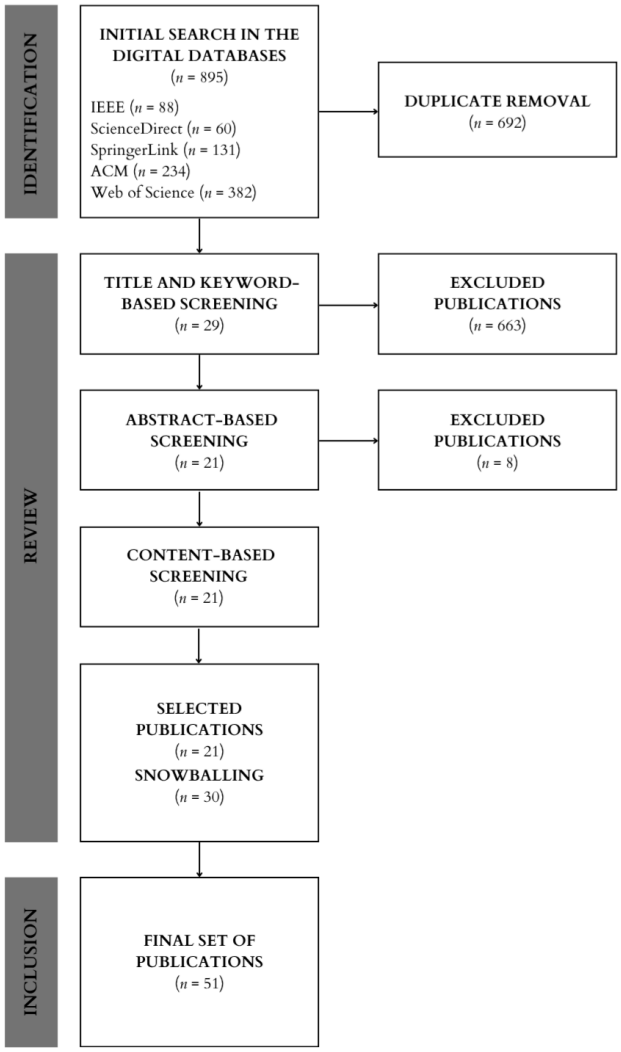


Figure 1: Systematic literature review process.

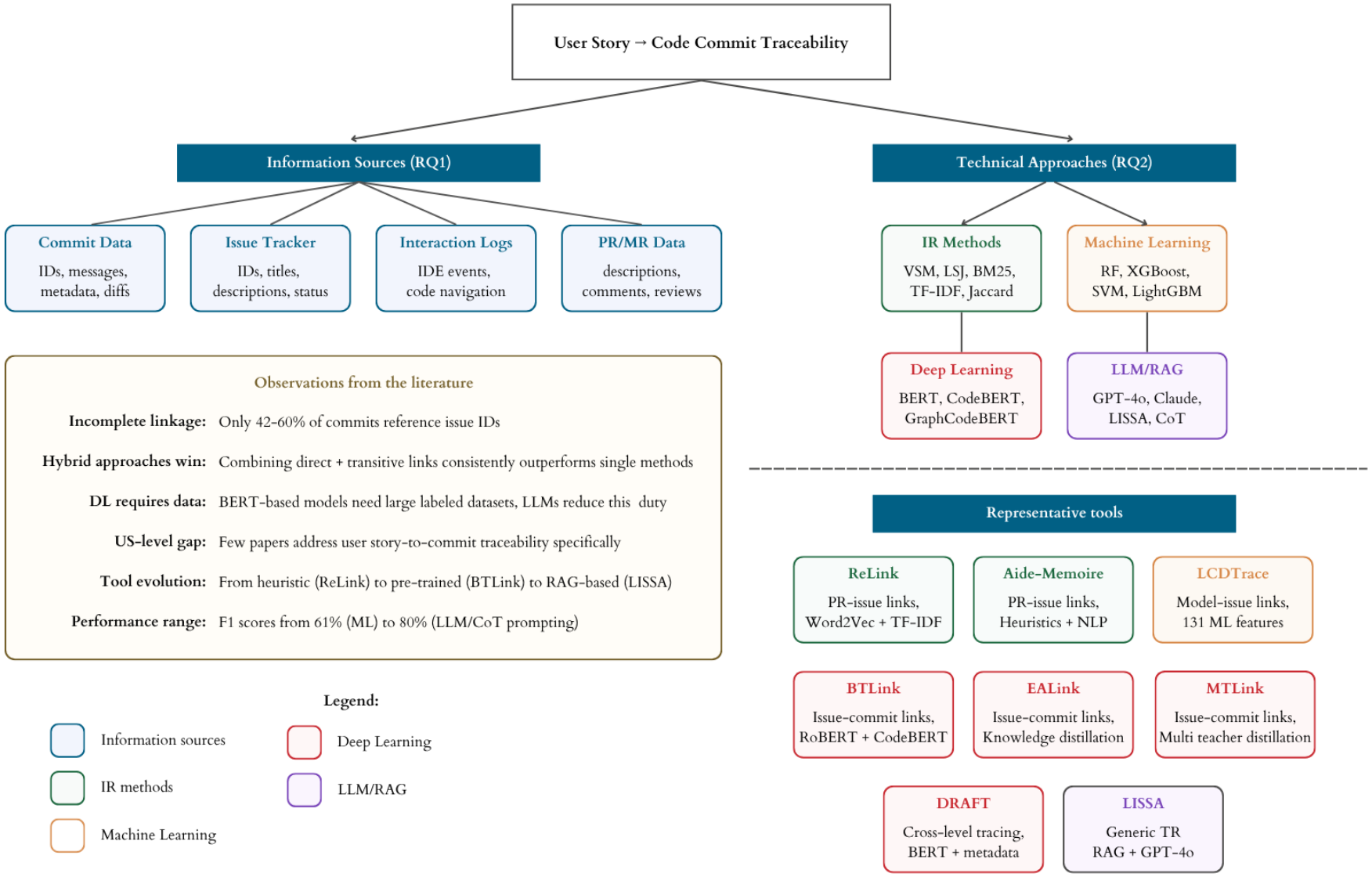


Figure 2: Identified traceability recovery techniques and tools.

Table 2: Identified literature included in the review.

ID	Title	Authors	Source
S01	What you see is what you trace: a two-stage interview study on traceability practices and eye tracking potential	Ahrens et al.	[13]
S02	Evaluating the Use of LLMs for Documentation to Code Traceability	Alor et al.	[14]
S03	PI-Link: A Ground-Truth Dataset of Links Between Pull-Requests and Issues in GitHub	Alshara et al.	[15]
S04	A Literature Review of Automatic Traceability Links Recovery for Software Change Impact Analysis	Aung et al.	[7]
S05	Cost and Benefit of Tracing Features with Embedded Annotations	Berger et al.	[16]
S06	Recovering from a Decade: A Systematic Mapping of Information Retrieval Approaches to Software Traceability	Borg et al.	[17]
S07	Locating requirements in backlog items: Content analysis and experiments with large language models	Can & Dalpiaz	[18]
S08	SoK: Systematizing Software Artifacts Traceability via Associations, Techniques, and Applications	Chen et al.	[6]
S09	MTLink: Adaptive multi-task learning based pre-trained language model for traceability link recovery between issues and commits	Deng et al.	[19]
S10	Enabling the Reuse of Software Development Assets Through a Taxonomy for User Stories	Ednaldo Dilorenzo et al.	[20]
S11	LiSSA: Toward Generic Traceability Link Recovery Through Retrieval-Augmented Generation	Fuchß et al.	[21]
S12	Beyond Retrieval: A Study of Using LLM Ensembles for Candidate Filtering in Requirements Traceability	Fuchß et al.	[22]
S13	Using Consensual Biterms from Text Structures of Requirements and Code to Improve IR-Based Traceability Recovery	Gao et al.	[23]
S14	Requirements Classification for Traceability Link Recovery	Hey et al.	[24]
S15	Improving Traceability Link Recovery Using Fine-grained Requirements-to-Code Relations	Hey et al.	[25]
S16	LLM-Based Multi-Agent Systems for Software Engineering: Literature Review, Vision, and the Road Ahead	He et al.	[26]
S17	Large Language Models for Software Engineering: A Systematic Literature Review	Hou et al.	[27]
S18	Is this Build Failure Related to my Patch? An Empirical Study of Unrelated Build Failures in Continuous Integration	Huang et al.	[28]
S19	Are your comments outdated? Towards automatically detecting code-comment consistency	Huang et al.	[29]
S20	Interaction-based creation and maintenance of continuously usable trace links between requirements and source code	Hübner & Paech	[4]
S21	An LLM-based Approach to Recover Traceability Links between Security Requirements and Goal Models	Jameleddine Hassine	[30]
S22	Mining architectural information: A systematic mapping study	Jean de Dieu et al.	[31]
S23	UserTrace: User-Level Requirements Generation and Traceability Recovery from Software Project Repositories	Jin et al.	[32]
S24	A Systematic Mapping Review: Tracking the Relationships Between Software Artifacts using NLP	Khalil et al.	[9]

Continued on next page

ID	Title	Authors	Source
S25	BTLink: automatic link recovery between issues and commits based on pre-trained BERT model	Lan et al.	[33]
S26	Traceability Transformed: Generating more Accurate Links with Pre-Trained BERT Models	Lin et al.	[34]
S27	Generating and visualizing trace link explanations	Liu et al.	[35]
S28	A Systematic Literature Review of Issue-Based Requirement Traceability	Lyu et al.	[8]
S29	On understanding and predicting issue links	Lüders et al.	[36]
S30	Automated Recovery of Issue-Commit Links Leveraging Both Textual and Non-textual Data	Mazrae et al.	[37]
S31	Automatic Traceability Maintenance via Machine Learning Classification	Mills et al.	[38]
S32	Improving the effectiveness of traceability link recovery using hierarchical bayesian networks	Moran et al.	[3]
S33	A systematic literature review of pre-requirements specification traceability	Mucha et al.	[39]
S34	Mining commit messages to enhance software refactorings recommendation: A machine learning approach	Nyamawe	[40]
S35	Natural Language Processing in Software Engineering: A Systematic Literature Review	Pacheco et al.	[10]
S36	Aide-mémoire: Improving a Project's Collective Memory via Pull Request–Issue Links	Pârțachi et al.	[2]
S37	User Stories and Natural Language Processing: A Systematic Literature Review	Raharjana et al.	[1]
S38	Prompts Matter: Insights and Strategies for Prompt Engineering in Automated Software Traceability	Rodriguez et al.	[41]
S39	Leveraging Intermediate Artifacts to Improve Automated Trace Link Retrieval	Rodriguez et al.	[42]
S40	Why don't we trace? A study on the barriers to software traceability in practice	Ruiz et al.	[5]
S41	From Bugs to Benefits: Improving User Stories by Leveraging Crowd Knowledge with CrUISE-AC	Schwedt & Ströder	[43]
S42	User stories as boundary objects in agile requirements engineering: A theoretical literature review	Sporsem et al.	[44]
S43	Analysis and comparison of deep learning models for user stories identification	Veitía et al.	[45]
S44	On the Impact of Requirements Smells in Prompts: The Case of Automated Traceability	Vogelsang et al.	[46]
S45	R2Code: A Self-Reflective LLM Framework for Requirements-to-Code Traceability	Wang et al.	[47]
S46	An Empirical Study on Data Balancing in Machine Learning Based Software Traceability Methods	Wang et al.	[48]
S47	DF4RT: Deep Forest for Requirements Traceability Recovery Between Use Cases and Source Code	Wang et al.	[49]
S48	Natural Language Processing Application on Commit Messages: A Case Study on HEP Software	Yang et al.	[50]
S49	Evaluating ReLink for Traceability Link Recovery in Practice	Yaşa et al.	[51]
S50	EALink: An Efficient and Accurate Pre-trained Framework for Issue-Commit Link Recovery	Zhang et al.	[52]

Continued on next page

Table 3
Information sources for linking user stories to commits

Source	Description	Advantage	Limitations
Commit message issue IDs	Explicit identifier	Direct, machine-readable	Only ~42–60% of commits include them [4, 37]
Issue tracker metadata	Title, description, assignee, labels	Rich semantic and contextual data	Requires access to the tracker
Developer interaction logs	IDE file selections, edits, timestamps	Captures uncommitted work context	Requires tool instrumentation
Pull request descriptions	Developer-written change summaries	High semantic fidelity	Only present in PR-based workflows

ID	Title	Authors	Source
S51	Enhancing Traceability Link Recovery with Unlabeled Data	Zhu et al.	[53]

4. Linking user stories to source code commits

4.1. Information sources for traceability links

Establishing traceability between user stories and source code commits in agile projects relies on several information sources that developers naturally produce during their workflow. The most common sources include commit messages, issue tracker metadata, developer interaction logs, and code change artifacts.

1. **Commit Messages and Issue Identifiers:** A common practice in agile development is to embed issue identifiers (e.g., Jira ticket IDs) in commit messages [34]. This practice creates a direct, machine-readable connection between version control systems and issue trackers. However, empirical evidence reveals significant gaps in this practice. Rath et al. found that only 60% of commits contain an issue ID across seven open-source projects [4]. Hübner & Paech further researched this gap, reporting that only 42.2% of issues were linked to corresponding commits in an analysis of over half a million GitHub issues [37]. These findings show that while commit messages are the most direct information source, they are far from universally accepted.
2. **Issue Tracker Metadata:** Beside the identifier itself, issue trackers such as Jira and GitHub Issues provide additional contextual data including issue titles, descriptions, assignees, labels, and similar information. This metadata has also been used for traceability in some cases. DRAFT [54], a deep learning approach, integrates metadata features alongside textual representations, to improve link accuracy. Similarly, MTLink [19] exploits code heuristics and metadata such as temporal overlap and contributor identity to recover issue-commit links.
3. **Developer Interaction Logs.** A less usual but promising information source is the recording of developer interactions within integrated development environments (IDEs). ILCom captures which source code files a developer selects and edits while working on a requirement, associating these interactions with the requirement's issue ID extracted from the commit message [4]. This approach achieved precision above 90% and recall of approximately 80%, demonstrating that interaction data has the ability to complement textual sources [4].
4. **Pull Request Descriptions:** In projects using GitHub-style workflows, pull request (PR) descriptions offer additional natural language context about the intent of code changes. Aide-memoire [2] exploits PR descriptions rather than relying on automatically generated change summaries, arguing that developer-written descriptions are more semantically faithful to the actual intent. The PI-Link study established ground-truth datasets of PR-issue links to support research in this area [15].

A consolidated list of the information source, their description, advantages and limitations is presented in Table 3.

4.2. Approaches for Establishing Traceability Links

Research on traceability link recovery has evolved through several methodological generations, from traditional information retrieval to deep learning and large language model (LLM) architectures.

Information Retrieval (IR) Methods: Early approaches applied algebraic models such as the Vector Space Model (VSM) and Latent Semantic Indexing (LSI) to calculate textual similarity between artifacts expressed in natural language [7]. Borg et al.'s systematic mapping identified 79 publications on IR-based trace recovery, noting the dominance of these techniques [1]. However, they are highly sensitive to project-specific parameter tuning, which limits their generalizability. ReLink, one of the earliest automated tools, measured similarity between change logs and bug reports using TF-IDF vectors and cosine similarity for true links [51].

Machine Learning (ML) Approaches. ML techniques introduced the ability to combine multiple features for link prediction. Rath et al. used random forest classifiers trained on textual similarity features, metadata (developer identity, temporal information), and code-level features to identify matching issues for unlinked commits [4]. TRAIL proposed a machine learning classifier trained on historically collected traceability information to classify links as valid or invalid, achieving effectiveness across both new and changed artifacts [38].

Deep Learning and Pre-trained Models. The adoption of transformer-based pre-trained models has marked a significant advancement. T-BERT (Trace BERT) applies a three-step training strategy, using transfer learning to generate trace links between source code and natural language artifacts [34]. BTLink combines RoBERTa (for issue text) and CodeBERT (for code) through a fusion layer and classifier, achieving state-of-the-art performance across all evaluation metrics in both within-project and cross-project settings [33]. EALink introduces knowledge distillation from large models to create compact student models, reducing computational overhead while maintaining accuracy improvements of 1–6% across large-scale Apache projects [52]. MTLink employs multi-teacher knowledge distillation with an adaptive multi-task learning strategy to recover issue-commit links [19].

Retrieval-Augmented Generation (RAG) and LLMs: The most recent approaches incorporate large language models. LiSSA uses embedding-based retrieval followed by GPT-4o classification with Chain-of-Thought prompting, significantly outperforming prior methods like FTLR and COMET for requirement-to-code tracing [21]. Beyond Retrieval evaluates ensembles of small language models (<15B parameters) to replace traditional IR for candidate filtering, demonstrating that LLM chaining effectively reduces the search space while maintaining high recall [22].

4.3. What information is most effective?

The literature reveals that **no** single information source or technique is **universally superior**. Rather, effectiveness depends on the combination of features and the project context. *Textual content remains foundational* - even the most advanced models rely on the textual similarity between issue descriptions and commit messages or code changes as a baseline signal. *Metadata provides critical supplementary data:* DRAFT's [54] integration of 11 metadata features alongside BERT embeddings achieved average F1 and F2 scores of 69.3% and 76.9%, respectively, showing that metadata features complement textual representations. Hybrid-Linker [37] leverages both textual and non-textual data, showing that combining commit message text, code changes, and project metadata outperforms text-only approaches. And finally, *multi-task and cross-artifact approaches improve robustness.* ILCom [4] shows that combining issue IDs from commit messages with developer interaction logs in the IDE achieves precision above 90%. MTLink's [19] multi-task learning strategy, incorporating issue-commit, issue-code, and one-to-many link recovery, showed that modeling multiple artifact relationships simultaneously enhances link quality. Table 4 presents the representative approaches mentioned, as well as a comparison of their documented

effectiveness. This comparison should not be interpreted as directly comparable, as the papers use different datasets, experimental settings and evaluation methodologies.

Table 4
Representative approaches and their effectiveness

Approach	Method	Key Features	Performance	Source
ReLink	IR + heuristics	TF–IDF cosine similarity, metadata	Top-5 accuracy: 0.80, MRR: 0.84	[51]
ILCom	Interaction recording	IDE interactions + issue IDs	Precision: 90%, Recall: 79%	[4]
T-BERT	Pre-trained BERT	Transfer learning from code tasks	Outperforms IR baselines significantly	[34]
BTLink	RoBERTa + CodeBERT	Dual-encoder fusion	State-of-the-art across all metrics	[33]
DRAFT	BERT + metadata	11 heuristic features	F1: 69.3%, F2: 76.9%	[54]
LiSSA	RAG + LLM	GPT-4o with Chain-of-Thought	Outperforms FTLR and COMET	[21]
EALink	Knowledge distillation	Contrastive + multi-task learning	1–6% accuracy improvement	[52]

In summary, the most effective strategies combine multiple information sources (textual content, metadata, interaction data, and code changes) within architectures that can bridge the semantic gap between natural language user stories and the actual code base. The field is moving toward multi-modal, LLM-powered approaches, but the primary importance of explicit issue identifiers in commit messages remains unchallenged.

5. Techniques for establishing traceability

The transition from manual to automated traceability practices is motivated by the increasing size and complexity of modern software systems, where manual maintenance is often impractical [32]. Due to the labor-intensive nature and high error rates of manual linking, which commonly lead to "traceability debt," it causes a variety of computational methods to reconstruct these associations that are present in the literature. This section reviews the technical foundations, ranging from traditional information retrieval to advanced deep learning, and identifies the specialized tools developed to implement these techniques in agile environments.

5.1. Technical approaches for trace link recovery

Automated traceability link recovery (TLR) techniques aim to bridge the semantic gap between high-level natural language artifacts (user stories) and low-level programming language artifacts (commits/source code) [25].

1. **Information Retrieval (IR) and Fuzzy Matching**
   Traditional IR methods treat traceability as a retrieval task based on textual similarity [32]. Common algebraic models include the *Vector Space Model (VSM)* and *Latent Semantic Indexing (LSI)*, which represent artifacts as term-based vectors and compute similarity scores through statistical co-occurrence patterns [32, 7]. While computationally efficient, these methods suffer from the "vocabulary mismatch problem," where different terms denote the same concept across concept levels [23, 47]. Fuzzy logic and approximate matching are used to control ambiguity and imprecision in requirements [10]. For instance, *fuzzy set theory and fuzzy lattice reasoning* have been used to classify the membership of artifacts in a trace set when information is incomplete [1, 7]. Advanced string-level metrics and Jaccard's similarity are often used as "fuzzy" baselines to complement more complex models [55, 10].

2. **Natural Language Processing (NLP)**
   NLP is fundamental for interpreting the natural language within user stories. Techniques such as *Named Entity Recognition (NER)*, *part-of-speech (POS) tagging*, and *semantic role labeling* are used to identify critical entities and their roles (e.g., "who", "what", "where") within requirements [1, 14]. Modern NLP approaches incorporate word embeddings (e.g., Word2Vec, GloVe) and *Word Mover's Distance (WMD)* to compute similarities based on semantic relatedness rather than mere syntactical overlap [25].
3. **Machine Learning (ML)**
   Supervised ML approaches formulate TLR as a binary classification problem, training models on historically verified links to predict the validity of new candidate links [38, 14]. Common algorithms include *Random Forests, Support Vector Machines (SVM)*, and *Naïve Bayes* [9, 8]. More advanced probabilistic models, such as *Hierarchical Bayesian Networks*, can simultaneously model transitive relationships and incorporate developer feedback to improve inference [3].
4. **Deep Learning (DL) and Transformers**
   Recent advancements leverage pre-trained language models like *BERT, RoBERTa*, and *CodeBERT* to generate contextualized embeddings [41]. Architectures like *BTLink* use dual encoders to analyze issue text and commit code separately before a fusion layer identifies links [30]. Frameworks such as *T-BERT* (Trace BERT) use transfer learning from closely related software engineering challenges to overcome data sparsity and achieve state-of-the-art accuracy [34]. *EALink* and *MTLink* introduce knowledge distillation to create efficient student models, reducing the computational overhead of large-scale transformers while maintaining high performance [52, 19].

A brief comparison of the technical approaches presented in this subsection are shown in Table 5.

Table 5
Comparison of Technical Approaches

Category	Techniques	Semantic depth	Limitations
IR/Fuzzy	VSM, LSI, TF-IDF, Fuzzy Logic	Low (Keyword-based)	Vocabulary mismatch, lacks context
Machine learning	Random Forest, SVM, Bayesian Networks	Moderate (feature-based)	Requires large labeled datasets
Deep learning	BERT, CodeBERT, T-BERT, LSTMs	High (Contextual)	High computational cost and data sparsity
LLM/RAG	GPT-4o, Prompt Engineering, CoT	Very High (Reasoning)	Sensitive to prompt framing, token cost

5.2. Automated tools and implementations

Several tools have been developed to implement the approaches presented in Section 5.1, each with distinct functionalities and usage scenarios. In Table 6 we present a brief overview of the representative tools that could be found in our literature set.

5.3. Limitations and implementation challenges

Despite certain advancements, several challenges still persist when applied in practical settings. High precision is consistently reported by recent LLM-based approaches (87%–100%), but recall remains variable (47%–75%), often restricted by artifact detail [14]. In the same reference, authors also report common error patterns, which include:

- **Naming-based assumptions (IAE):** Incorrect links formed solely due to similar identifiers.
- **Phantom links (PAL):** Linking to non-existent code elements.
- **Over-generalization (APB):** Applying architectural patterns inaccurately across artifacts.

Table 6
Analysis of representative traceability tools

Tool	Technical base	Functionality	Limitations
ReLink	IR + Heuristics	Offline recovery of missing bug-commit links	Heuristic accuracy and manual vetting
Aide-mémoire	Online ML (Mondrian Forest)	Real-time PR-to-issue link suggestions	Low recall (30%), requires bootstrapping
TRAIL	ML Classification	Automatic maintenance of evolving links	Sensitive to class imbalance (traces vs no-traces)
LiSSA	RAG + LLM (GPT-4o)	Generic recovery across heterogeneous artifacts	Defining links for LLM interpretation can be ambiguous
FTLR	Unsupervised word embeddings	Fine-grained requirement-to-code tracing	Unable to attain full recall
UserTrace	Multi-agent LLM	Generating user-level requirements and live links	Computationally intensive for large repositories

- **Implementation overlinking (IOL):** Linking high-level requirements to excessively granular internal code details.

Furthermore, industrial adoption is challenged by the manual-intensive nature of manual link maintenance, often leading to "traceability debt" [6, 13]. Most tools remain at the proof-of-concept stage, with very few being publicly available or integrated into everyday developer workflows [56].

6. Discussion

This paper has examined the state of the art in connecting user stories to source code commits, a task that is needed to maintain transparency and quality in agile software development.

RQ1: How can user stories be linked to source code commits? Linkage is typically established through both direct and indirect approaches. Direct methods rely on the semantic similarity between the natural language of user stories and the identifiers or comments found in source code. Indirect methods, which often result in higher accuracy in complex systems, leverage intermediate artifacts such as issue tracking tickets, architectural models, and commit metadata (e.g., authorship, timestamps) as bridges between the high-level requirements and the implementation.

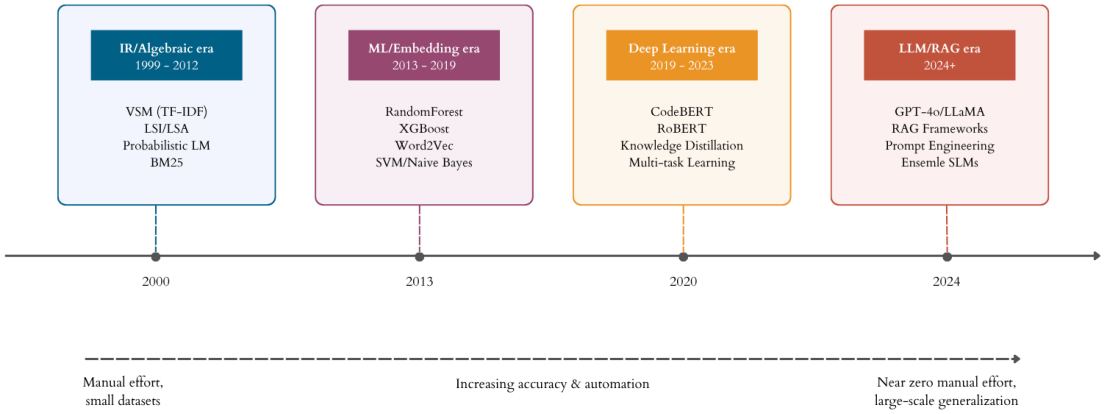


Figure 3: Evolution of traceability techniques.

RQ2: What techniques exist for establishing traceability? The field has transitioned through several technological waves (see Figure 3). Early approaches utilized algebraic Information Retrieval (IR) models like the Vector Space Model (VSM) and Latent Semantic Indexing (LSI). These were subsequently augmented by Machine Learning (ML) techniques, including Random Forests and word embeddings (Word2Vec), which incorporated metadata heuristics to improve precision.

In recent years, the focus has shifted toward Deep Learning and Large Language Models (LLMs). Pre-trained transformer models, such as BERT and CodeBERT, have become the new standard, exemplified by frameworks like BTLink and DRAFT. The current state-of-the-art is represented by Retrieval-Augmented Generation (RAG) systems like LiSSA and efficient multi-teacher distillation models like MTLink. These architectures utilize the reasoning capabilities of LLMs to achieve high accuracy with minimal manual intervention, marking a significant step toward fully automated and generalized traceability in modern software engineering.

7. Conclusion and future work

This SLR has provided a synthesis of the state-of-the-art in establishing traceability between user stories and source code commits within agile software development. By analyzing a total of 51 primary studies, the research shows a clear technological trajectory: the field has evolved from traditional Information Retrieval models, such as Vector Space Models (VSM) and Latent Semantic Indexing (LSI), toward sophisticated Machine Learning approaches using embeddings and metadata heuristics, and more recently to state-of-the-art Deep Learning and Large Language Model (LLM) architectures. The paper highlights how modern frameworks leverage knowledge distillation and Retrieval-Augmented Generation (RAG) to achieve high-precision tracing with minimal manual intervention, marking a significant shift from earlier techniques that often required project-specific tuning.

An important contribution of this work is the identification of a specialized focus on the Git ecosystem, where issues, pull requests, and commits serve as the primary artifacts for bridging the gap between requirements and implementation. The findings establish traceability between user stories and source code commits as a key prerequisite for user story mining, offering guidance for researchers and practitioners interested in automated requirements analysis and story-to-code reconstruction using large-scale open-source repositories.

Building on the findings of this review, our future research will focus on selecting and extending one of the identified traceability approaches to develop a tool that supports effort estimation in agile software development. The proposed tool will use historical Git repository data to automatically link user stories (or other agile work items) with source code commits, allowing the extraction of historical information that can be used to improve data-driven effort estimation models.

Acknowledgments

The authors acknowledge financial support from the Slovenian Research and Innovation Agency (Research Core Funding No. P2-0057).

Declaration on Generative AI

During the preparation of this work, the authors used Claude, ChatGPT and Grammarly in order to: translate text, paraphrase and reword, check grammar and spelling. After using these tools and services, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] I. K. Raharjana, D. Siahaan, C. Fatichah, User stories and natural language processing: A systematic literature review, IEEE Access 9 (2021) 53811–53826. doi:10.1109/ACCESS.2021.3070606.

[2] P.-P. Pârţachi, D. R. White, E. T. Barr, Aide-mémoire: Improving a project's collective memory via pull request–issue links, ACM Trans. Softw. Eng. Methodol. 32 (2023). URL: https://doi.org/10.1145/3542937. doi:10.1145/3542937.

[3] K. Moran, D. N. Palacio, C. Bernal-Cárdenas, D. McCrystal, D. Poshyvanyk, C. Shenefiel, J. Johnson, Improving the effectiveness of traceability link recovery using hierarchical bayesian networks, in: Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering, ICSE '20, Association for Computing Machinery, New York, NY, USA, 2020, p. 873–885. URL: https://doi.org/10.1145/3377811.3380418. doi:10.1145/3377811.3380418.

[4] P. Hübner, B. Paech, Interaction-based creation and maintenance of continuously usable trace links between requirements and source code, Empirical Software Engineering 25 (2020) 4350–4377. URL: https://doi.org/10.1007/s10664-020-09831-w. doi:10.1007/s10664-020-09831-w.

[5] M. Ruiz, J. Y. Hu, F. Dalpiaz, Why don't we trace? a study on the barriers to software traceability in practice, Requirements Engineering 28 (2023) 619–637. URL: https://doi.org/10.1007/s00766-023-00408-9. doi:10.1007/s00766-023-00408-9.

[6] Z. Chen, L. Yi, L. Nie, Y. Zhao, H. Liu, Y. Shi, W. Song, Sok: Systematizing software artifacts traceability via associations, techniques, and applications, 2026. URL: https://arxiv.org/abs/2603.16208. arXiv:2603.16208.

[7] T. W. W. Aung, H. Huo, Y. Sui, A literature review of automatic traceability links recovery for software change impact analysis, in: Proceedings of the 28th International Conference on Program Comprehension (ICPC), 2020, pp. 14–24. doi:10.1145/3387904.3389251.

[8] Y. Lyu, H. Cho, P. Jung, S. Lee, A systematic literature review of issue-based requirement traceability, IEEE Access 11 (2023) 13334–13348. doi:10.1109/ACCESS.2023.3242294.

[9] F. Khalil, G. Rebdawi, N. Ghneim, A systematic mapping review: Tracking the relationships between software artifacts using nlp, ECTI Transactions on Computer and Information Technology (ECTI-CIT) 19 (2025) 321–333. doi:10.37936/ecti-cit.2025192.258621.

[10] G. Pacheco, L. Martins, A. Silva, N. Lavesson, T. Gorschek, Natural language processing in software engineering: A systematic literature review, Journal of Software Engineering Research and Development 13 (2025). doi:10.5753/jserd.2025.5097.

[11] S. B. A. Rizvi, Z. Zhang, A SYSTEMATIC MAPPING STUDY ON THE USE OF LLMS FOR REQUIREMENTS TRACEA-BILITY LINK DISCOVERY, Ph.D. thesis, 2025. URL: https://trepo.tuni.fi/bitstream/handle/10024/229538/RizviSyedBabarAli.pdf?sequence=2&isAllowed=y.

[12] B. Kitchenham, O. Pearl Brereton, D. Budgen, M. Turner, J. Bailey, S. Linkman, Systematic literature reviews in software engineering – a systematic literature review, Information and Software Technology 51 (2009) 7–15. URL: https://www.sciencedirect.com/science/article/pii/S0950584908001390. doi:https://doi.org/10.1016/j.infsof.2008.09.009, special Section - Most Cited Articles in 2002 and Regular Research Papers.

[13] M. Ahrens, L. Nagel, K. Schneider, What you see is what you trace: a two-stage interview study on traceability practices and eye tracking potential, Requirements Engineering 29 (2024) 281–302. doi:10.1007/s00766-024-00419-0.

[14] E. Alor, S. Khatoonabadi, E. Shihab, Evaluating the use of llms for documentation to code traceability, arXiv preprint (2025).

[15] Z. Alshara, A. Shatnawi, H. Eyal-Salman, A.-D. Seriai, M. Shatnawi, Pi-link: A ground-truth dataset of links between pull-requests and issues in github, IEEE Access 10 (2022) 127446–127457. doi:10.1109/ACCESS.2022.3232982.

[16] T. Berger, W. Mahmood, R. A. Zahra, I. Vassilevski, A. Burger, W. Ji, M. Antkiewicz, K. Czarnecki, Cost and benefit of tracing features with embedded annotations, ACM Transactions on Software Engineering and Methodology (2025). doi:10.1145/3746060.

[17] M. Borg, P. Runeson, A. Ardö, Recovering from a decade: A systematic mapping of information

retrieval approaches to software traceability, Empirical Software Engineering 19 (2013) 1568–1603. doi:10.1007/s10664-013-9255-y.

[18] A. T. van Can, F. Dalpiaz, Locating requirements in backlog items: Content analysis and experiments with large language models, Information and Software Technology 174 (2024) 107509. doi:10.1016/j.infsof.2024.107509.

[19] Y. Deng, B. Wang, Q. Zhu, J. Liu, J. Kuang, X. Li, Mtlink: Adaptive multi-task learning based pre-trained language model for traceability link recovery between issues and commits, Journal of King Saud University - Computer and Information Sciences 36 (2024) 101958. doi:10.1016/j.jksuci.2024.101958.

[20] E. Dilorenzo, E. Dantas, M. Perkusich, F. Ramos, A. Costa, D. Albuquerque, H. Almeida, A. Perkusich, Enabling the reuse of software development assets through a taxonomy for user stories, IEEE Access 8 (2020) 107285–107300. doi:10.1109/ACCESS.2020.2996951.

[21] D. Fuchß, T. Hey, J. Keim, H. Liu, N. Ewald, T. Thirolf, A. Koziolek, Lissa: Toward generic traceability link recovery through retrieval-augmented generation, in: Proceedings of the 47th IEEE/ACM International Conference on Software Engineering (ICSE), 2025, pp. 1396–1408. doi:10.1109/ICSE55347.2025.00186.

[22] D. Fuchß, S. Schwedt, J. Keim, T. Hey, Beyond retrieval: A study of using llm ensembles for candidate filtering in requirements traceability, in: Proceedings of the 33rd IEEE International Requirements Engineering Conference Workshops (REW), 2025, pp. 5–12. doi:10.1109/REW66121.2025.00006.

[23] H. Gao, H. Kuang, K. Sun, X. Ma, A. Egyed, P. Mäder, G. Rong, D. Shao, H. Zhang, Using consensual biterms from text structures of requirements and code to improve ir-based traceability recovery, in: Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2022, pp. 1–12. doi:10.1145/3551349.3556948.

[24] T. Hey, J. Keim, S. Corallo, Requirements classification for traceability link recovery, in: Proceedings of the 32nd IEEE International Requirements Engineering Conference (RE), 2024, pp. 155–167. doi:10.1109/RE59067.2024.00024.

[25] T. Hey, F. Chen, S. Weigelt, W. F. Tichy, Improving traceability link recovery using fine-grained requirements-to-code relations, in: Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE), 2021, pp. 1189–1200. doi:10.1145/3468264.3473122.

[26] J. He, C. Treude, D. Lo, Llm-based multi-agent systems for software engineering: Literature review, vision, and the road ahead, ACM Transactions on Software Engineering and Methodology (2025). doi:10.1145/3712003.

[27] X. Hou, Y. Zhao, Y. Liu, Z. Yang, K. Wang, L. Li, X. Luo, D. Lo, J. Grundy, H. Wang, Large language models for software engineering: A systematic literature review, ACM Transactions on Software Engineering and Methodology (2023). doi:10.1145/3695982.

[28] A. Huang, D. A. da Costa, G. Dick, M. E. Mezouar, Is this build failure related to my patch? an empirical study of unrelated build failures in continuous integration, Empirical Software Engineering (2026).

[29] Y. Huang, Y. Chen, X. Chen, X. Zhou, Are your comments outdated? towards automatically detecting code-comment consistency, Empirical Software Engineering 29 (2024) 57. doi:10.1007/s10664-024-10448-4.

[30] J. Hassine, An llm-based approach to recover traceability links between security requirements and goal models, in: Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering (EASE), 2024, pp. 643–651. doi:10.1145/3661167.3661261.

[31] M. J. de Dieu, P. Liang, M. Shahin, C. Yang, Z. Li, Mining architectural information: A systematic mapping study, Empirical Software Engineering 29 (2024) 82. doi:10.1007/s10664-024-10480-6.

[32] D. Jin, Z. Jin, Y. Zhang, Z. Fang, L. Li, Y. He, X. Chen, W. Sun, Usertrace: User-level requirements generation and traceability recovery from software project repositories, ArXiv (2025).

[33] J. Lan, L. Gong, J. Zhang, H. Zhang, Btlink: automatic link recovery between issues and commits

based on pre-trained bert model, volume 28, 2023, p. 103. doi:10.1007/s10664-023-10342-7.
[34] J. Lin, Y. Liu, Q. Zeng, M. Jiang, J. Cleland-Huang, Traceability transformed: Generating more accurate links with pre-trained bert models, in: Proceedings of the 43rd IEEE/ACM International Conference on Software Engineering (ICSE), 2021, pp. 324–335. doi:10.1109/ICSE43902.2021.00040.
[35] Y. Liu, J. Lin, O. Anuyah, R. Metoyer, J. Cleland-Huang, Generating and visualizing trace link explanations, in: Proceedings of the 44th International Conference on Software Engineering (ICSE), 2022, pp. 1471–1482. doi:10.1145/3510003.3510129.
[36] C. M. Lüders, T. Pietz, W. Maalej, N. D. Palo, E. Johns, On understanding and predicting issue links, Requirements Engineering 28 (2023) 339–358. doi:10.1007/s00766-023-00406-x.
[37] P. R. Mazrae, M. Izadi, A. Heydarnoori, Automated recovery of issue-commit links leveraging both textual and non-textual data, in: Proceedings of the 18th International Conference on Mining Software Repositories (MSR), 2021, pp. 483–494. doi:10.1109/MSR52588.2021.00062.
[38] C. Mills, J. Escobar-Avila, S. Haiduc, Automatic traceability maintenance via machine learning classification, in: Proceedings of the 28th International Conference on Program Comprehension (ICPC), 2020, pp. 244–255. doi:10.1145/3387904.3389309.
[39] J. Mucha, A. Kaufmann, D. Riehle, A systematic literature review of pre-requirements specification traceability, Requirements Engineering 29 (2024) 119–141. doi:10.1007/s00766-023-00412-z.
[40] A. S. Nyamawe, Mining commit messages to enhance software refactorings recommendation: A machine learning approach, Machine Learning with Applications 10 (2022) 100316. doi:10.1016/j.mlwa.2022.100316.
[41] A. D. Rodriguez, K. R. Dearstyne, J. Cleland-Huang, Prompts matter: Insights and strategies for prompt engineering in automated software traceability, in: Proceedings of the 31st IEEE International Requirements Engineering Conference Workshops (REW), 2023, pp. 455–464. doi:10.1109/REW57809.2023.00087.
[42] A. D. Rodriguez, J. Cleland-Huang, D. Falessi, Leveraging intermediate artifacts to improve automated trace link retrieval, in: Proceedings of the 2021 IEEE International Conference on Software Maintenance and Evolution (ICSME), 2021, pp. 471–482. doi:10.1109/ICSME52107.2021.00014.
[43] S. Schwedt, T. Ströder, From bugs to benefits: Improving user stories by leveraging crowd knowledge with cruise-ac, in: Proceedings of the 47th International Conference on Software Engineering (ICSE), 2025.
[44] T. Sporsem, T. Dingsøyr, K.-J. Stol, User stories as boundary objects in agile requirements engineering: A theoretical literature review, Journal of Systems and Software 215 (2025) 112093. doi:10.1016/j.jss.2025.112093.
[45] F. J. P. Veitía, M. L. Roldán, M. M. Vegetti, Analysis and comparison of deep learning models for user stories identification, Ingeniare. Revista chilena de ingeniería 32 (2024) 229–242. doi:10.4067/s0718-33052023000100229.
[46] A. Vogelsang, A. Korn, G. Broccia, A. Ferrari, J. Fischbach, C. Arora, On the impact of requirements smells in prompts: The case of automated traceability, in: Proceedings of the 47th International Conference on Software Engineering (ICSE), 2025.
[47] Y. Wang, J. Keung, X. Ma, Z. Mao, K. Chen, Y. Li, R2code: A self-reflective llm framework for requirements-to-code traceability, ArXiv (2026).
[48] B. Wang, Y. Deng, H. Wan, X. Li, An empirical study on data balancing in machine learning based software traceability methods, 2023. doi:0.1109/IJCNN54540.2023.10191386.
[49] B. Wang, Y. Deng, H. Wan, X. Li, Df4rt: Deep forest for requirements traceability recovery between use cases and source code, in: Proceedings of the 2023 IEEE International Conference on Systems, Man, and Cybernetics (SMC), 2023, pp. 617–622. doi:10.1109/SMC53992.2023.10394259.
[50] Y. Yang, E. Ronchieri, M. Canaparo, Natural language processing application on commit messages: A case study on hep software, Applied Sciences 12 (2022) 10773. doi:10.3390/app122110773.
[51] A. Yaşa, C. K. Özaltan, G. Ayten, F. Kaplama, Ömercan Devran, B. M. Uçar, E. Tüzün, Evaluating relink for traceability link recovery in practice, in: Proceedings of the 32nd IEEE International

Conference on Software Analysis, Evolution and Reengineering (SANER), 2025, pp. 80–90. doi:10.1109/SANER64311.2025.00016.

[52] C. Zhang, Y. Wang, Z. Wei, Y. Xu, J. Wang, H. Li, R. Ji, Ealink: An efficient and accurate pre-trained framework for issue-commit link recovery, in: Proceedings of the 38th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2023, pp. 217–229. doi:10.1109/ASE56229.2023.00059.

[53] J. Zhu, G. Xiao, Z. Zheng, Y. Sui, Enhancing traceability link recovery with unlabeled data, in: Proceedings of the 33rd IEEE International Symposium on Software Reliability Engineering (ISSRE), 2022, pp. 446–457. doi:10.1109/ISSRE55802.2022.00051.

[54] J. Tian, L. Zhang, X. Lian, A cross-level requirement trace link update model based on bidirectional encoder representations from transformers, Mathematics 11 (2023) 623. doi:10.3390/math11030623.

[55] M. Abbas, A. Ferrari, A. Shatnawi, E. Enoiu, M. Saadatmand, D. Sundmark, On the relationship between similar requirements and similar software, Requirements Engineering 27 (2022) 63–80. doi:10.1007/s00766-021-00370-4.

[56] L. Zhao, W. Alhoshan, A. Ferrari, K. J. Letsholo, M. A. Ajagbe, E.-V. Chioasca, R. T. Batista-Navarro, Natural language processing for requirements engineering, ACM Computing Surveys 54 (2021) 1–38. doi:10.1145/3444689.