JOURNAL OF APPLIED SCIENCES TECHNOLOGY AND COMPUTING
e-ISSN: 3036-0250

JASTEC

Vol. 2 No. 1 (2025) 32-42
https://publisher.uthm.edu.my/ojs/index.php/jastec

Exploring Traceability Techniques on Software Engineering: A Review and Future Directions

Danish Hossman Abd Rahman[1], Rabatul Aduni Sulaiman[1]*, Misslina Jelani[2]

[1] *Faculty of Computer Science and Information Technology, Universiti Tun Hussein Onn Malaysia, 84600 Parit Raja, Johor, MALAYSIA*

[2] *Research Development Unit, Sekolah Menengah Kebangsaan Benut, Jalan Jaafar, 82200 Benut, Pontian, Johor, MALAYSIA*

*Corresponding Author: danishhossman73@gmail.com
DOI: https://doi.org/10.30880/jastec.2025.02.01.003

Article Info

Received: 19 March 2025
Accepted: 20 May 2025
Available online: 30 June 2025

Keywords

Traceability, modeling techniques, tool integration, blockchain, artificial intelligence, software engineering.

Abstract

This review explores innovative approaches to software traceability, emphasizing modeling techniques, tool integration, and future research directions. Software traceability, essential for tracking artifacts across development stages, enhances quality and stakeholder collaboration. The paper reviews methods such as formal models, graph-based approaches, blockchain solutions, and AI-assisted tools, comparing their advantages and limitations. Empirical studies show diverse strengths in scalability, usability, and accuracy, while highlighting challenges like standardization and stakeholder engagement. Future directions underscore the potential of integrating blockchain and artificial intelligence to develop holistic, scalable frameworks that address current limitations. Overall, emerging techniques promise to significantly advance traceability efficacy, with ongoing research needed to overcome practical challenges and improve adoption.

1. Introduction

Software traceability is broadly defined as the capacity to establish and maintain the relationships between various software artifacts throughout all phases of the software development lifecycle, including requirements, design, implementation, testing, and maintenance. According to [1], this foundational capability underpins critical engineering practices such as change-impact analysis, verification and validation, requirements satisfaction, and streamlined quality assurance. The concept of traceability not only encompasses the establishment of direct or indirect links across software artifacts but also facilitates navigating the complexity inherent in modern software ecosystems.

According to [1], the absence of fundamental traceability links particularly between key artifacts such as requirements and test cases that can delay effective software quality improvement and problem resolution. Their industrial experience indicates that missing or poorly maintained links often become barriers to resolving issues in complex and evolving projects. This challenge is evident in both commercial systems and large-scale academic development environments.

Similarly, [2] highlight the educational impact of structured traceability mechanisms. They demonstrate that different software development processes influence team productivity, cohesion, and overall software quality. The consistent use and enforcement of traceability practices, especially within iterative development settings, enhance process visibility, artifact consistency, and project outcomes.

In modern development contexts, traceability extends beyond traditional requirements-to-code mapping. For example, [3] proposes the use of execution traces in business process models to bridge linguistic gaps and expand traceability coverage among heterogeneous artifacts. Similarly, [4] introduces advanced machine learning

techniques to improve requirements-to-code traceability, showing that robust traceability models contribute directly to greater prediction accuracy and system maintainability. These examples highlight the diverse roles traceability assumes to serve as a navigational map across evolving project landscapes, a basis for automated compliance and assurance, and a facilitator for more adaptive and scalable development practices.

1.1 Motivation and Importance of Traceability

The drive for robust traceability systems in software engineering is multifaceted. At its core, effective traceability is central to ensuring software quality by providing the evidentiary backbone for compliance with standards, facilitating audits, and streamlining defect detection and resolution [1]. In practical terms, a lack of traceability is frequently cited as a root cause for failed maintenance activities, rework costs, and deployment delays, particularly as systems scale in complexity and multitude of stakeholders [2].

Furthermore, traceability serves as an essential mechanism for managing the complexity of interrelated artifacts in contemporary development where requirements volatility, evolving regulations, and multifaceted dependencies are the norm. As [5] address the dimension of data quality, underscoring that traceability is critical for explicitly capturing, validating, and iterating on data quality requirements. Likewise, traceability facilitates stakeholder collaboration, especially in distributed or regulated environments where artifact histories must be both transparent and trustworthy as noted in [6]. Blockchain-based traceability mechanisms, for example, have been shown to enable decentralized, tamper-evident record-keeping, thus supporting trust and accountability among varied project participants.

Practical challenges such as aligning business process models with system-level requirements, the lack of linguistic cues in model artifacts [3], and gaps in automation platforms [8] further highlight the need for innovative traceability approaches. Tool support, lifecycle alignment, and semantic consistency are frequently cited as both motivators in establishing end-to-end traceability, driving research in advanced modeling techniques and AI-assisted solutions [4], [9].

1.2 Scope and Objectives

Given the transformative role that traceability plays in software engineering, the aim of this review is to present a detailed survey of recent innovations in modeling techniques, tool integration, and emerging future trends. This includes a comparative analysis of traditional and modern modeling frameworks, such as machine learning-driven traceability link recovery [4], semantic models, and blockchain-enabled solutions [6]. The review further seeks to illuminate the integration of traceability into contemporary DevOps workflows and automation platforms, drawing from empirical studies on marketplace tool adoption [8] and architectural mapping techniques [10].

Additionally, the objectives extend to evaluating practical relevance and existing challenges such as inter-tool operability, stakeholder alignment, and traceability scalability in large collaborative projects as documented in case studies and experience reports by [7] and [3]. By synthesizing insights from data quality management [5], empirical education research [2], and advanced automation efforts [9], [4], this review aims to provide an actionable roadmap for both researchers and practitioners pursuing robust, future-ready traceability solutions. Overall, this review provides a comprehensive synthesis that bridges modeling-based, tool-oriented, and automation-driven approaches to traceability, while highlighting research gaps and outlining future opportunities for developing integrated, intelligent traceability frameworks.

This review provides a comprehensive synthesis of recent advances in software traceability between 2022 and 2025, bridging modelling-based techniques, tool-integrated practices, and automation-driven approaches within a unified analytical framework. Unlike prior studies that focused narrowly on individual methods or domains, this paper integrates empirical findings across multiple technological perspectives ranging from machine learning and blockchain to ontology and semantic reasoning. It further proposes a practical roadmap that emphasizes the development of explainable link recovery models, interoperable data representations, and auditable provenance mechanisms. Collectively, these insights aim to strengthen the connection between academic innovation and industrial application, enabling more transparent, scalable, and trustworthy traceability systems for future software engineering practices.

2. Related Work

A growing body of literature systematically investigates the state of traceability in software engineering through surveys, conceptual frameworks, and empirical studies. A comparative assessment conducted by [2] examined the impact of iterative, sequential, and “hands-off” development approaches in student computing projects, revealing how process structure influences quality, teamwork, and traceability-related factors. Their metric-based analysis provided foundational insights into how team dynamics and process formalization might enable or inhibit

traceability, especially highlighting the role of iterative methods in productivity and cohesion despite the tradeoff with final product quality.

Beyond surveys, several frameworks have been advanced to address traceability, especially in contexts requiring secure and consistent artifact lineage. In their study, [6] developed a blockchain-based requirement traceability framework for systems engineering, enabling dual-level traceability and decentralized consensus in requirements modification histories. Their methodology emphasizes the role of immutable, distributed ledgers in fostering trust and stakeholder collaboration, which is particularly valuable for complex projects involving distributed participants.

Advancing the technical methods of traceability link recovery, a study by [4] introduced GA-XWCoDe, a hybrid model integrating XGBoost, Node2Vec embeddings, and genetic algorithms to optimize the prediction of requirements-to-code traceability links. Their empirical evaluation showed marked improvements (17–33% higher F1 scores) compared to prior state-of-the-art approaches, indicating the potential of machine learning ensemble methods for stable and scalable traceability performance. In a related comparative study, [11] benchmarked six state-of-the-art methods for requirements-to-code traceability link recovery across 13 datasets, clarifying which models (e.g., CRT, TAROT, FTLR, and Random Forests) performed best under different project scales and providing a comparative ground for future tool selection and improvement.

NLP and information-retrieval-based traceability also feature prominently in recent research. According to [12], recent work has mapped the contributions and limitations of NLP techniques in software traceability, identifying key methods adopted for recovering traceability links across software artifacts. Their systematic mapping quantifies the increasing momentum in NLP-enabled approaches while also spotlighting persistent problems such as linguistic ambiguity, scalability, and explainability. In a related study, [13] automated conceptual model generation from behavior-driven development scenarios using NLP, with expert evaluations reaffirming the utility of such automated approaches in supporting agile traceability.

Tool-driven approaches have gained traction for their promise in automation and integration with broader software ecosystems. For example, a study by [8] catalogued over 8,000 tools from the GitHub Marketplace, providing a comprehensive view of the state of practice in automation and highlighting divergences between industry tool usage and academic research focus. In their work, [14] explored the use of ChatGPT for generating system specifications, demonstrating how modern AI can automate aspects of requirements engineering, though still facing challenges in requirements traceability and verification.

Despite these advancements, several gaps and limitations remain. Standardization presents a recurring issue. As noted by [5], the inherent complexity in eliciting and managing multi-dimensional data quality requirements indicates the need for harmonized models and guidelines for requirements traceability. Similarly, [12] underscored the lack of standard frameworks for NLP-based traceability due to inconsistencies in language, data representations, and process integration across projects. These concerns reflect broader calls for more unified, explainable, and interoperable traceability frameworks across the software engineering life cycle.

Scalability remains a challenge, particularly as systems and data volumes grow. For instance, as demonstrated in [15], automated performance model learning in large microservices systems shows both effectiveness and limitations, pointing to the need for scalable, low-overhead traceability that does not erode system performance or increase maintenance complexity. According to [7], even iterative, model-based methods require continuous adaptation and extension to handle project heterogeneity and evolving requirements at scale.

Stakeholder engagement and practical integration represent a further persistent gap. Both [5] and [6] emphasized the difficulties in capturing diverse stakeholder requirements and synchronizing changes in multi-participant settings. The limitations of existing tools in supporting meaningful, actionable traceability for non-technical stakeholders were also echoed by [14] and [8], who found that automated solutions can struggle with verification, maintenance, and user agreement, especially when interfaces or explanations do not align with stakeholders' perspectives. The reviewed literature underscores that while traceability research has evolved toward intelligent and automated solutions, gaps in consistency, scalability, and real-world integration remain, setting the stage for the methodological approach that follows.

Taken together, the reviewed studies demonstrate that software traceability has evolved from manual and document-centric practices toward intelligent, model-driven, and automation-assisted solutions. The literature reveals clear progress in applying machine learning, natural language processing, and blockchain to enhance accuracy, scalability, and auditability. However, persistent gaps remain in standardization, stakeholder alignment, and real-world integration that particularly concerning explainability and cross-tool interoperability.

These converging trends highlight an ongoing shift toward hybrid approaches that combine semantic, algorithmic, and decentralized methods to achieve end-to-end traceability across diverse artifacts. To systematically capture and evaluate these advancements, the following section outlines the methodological framework adopted in this review, detailing the data sources, selection criteria, and synthesis strategy used to analyze contemporary traceability techniques.

3. Methodology

This study adopts a structured review methodology designed to systematically collect, filter, and synthesize recent research on software traceability published between 2022 and 2025. The overall process followed three main stages: (i) data collection, (ii) screening and eligibility, and (iii) synthesis and analysis. Searches were conducted across major databases including IEEE Xplore, ACM Digital Library, ScienceDirect, and Scopus using combinations of the keywords "software traceability," "requirements-to-code," "blockchain, "automation," and "NLP". Inclusion criteria limited papers to peer-reviewed English-language journal articles that proposed or evaluated concrete traceability techniques or tools. Exclusion criteria omitted non-archival preprints, conceptual essays, and unrelated software management topics. Figure 1 illustrates the methodological framework that guided the review.

Fig. 1 *Methodological framework of the study*

Traceability in software engineering relies heavily on robust modeling techniques that support accurate, scalable, and maintainable links across artifacts. Among the most prominent approaches identified in the reviewed studies are formal methods, graph-based models, and blockchain-enabled frameworks, each offering unique strengths and challenges.

Formal methods, exemplified by modern traceability research, apply mathematically rigorous models and algorithms to ensure fidelity and correctness in linking requirements to implementation. As shown by [4], the GA-XWCoDe model integrates XGBoost and Node2Vec with genetic algorithms to optimize requirements-to-code traceability prediction. Their results show marked improvements in F1 scores and robustness across varying data scales, underscoring the advantage of ML-powered formalism in handling system complexity and adaptivity. Similarly, [16] leveraged code augmentation and fine-tuning of pre-trained language models (MS-CodeBERT) to perform multi-type requirements traceability prediction, demonstrating that data-driven formalization can directly improve precision and recall in real-world systems.

Graph-based models serve as another foundational modeling technique, especially for encapsulating relationships among software entities**.** As described by [10] and [3], graph-based approaches utilize static code analysis, version control data, and advanced graph traversal algorithms to capture dependencies and recommend optimal traceability links. Feature-to-code mappings, code dependency heatmaps, and visualized evolution histories not only highlight the interconnectedness of artifacts but also provide actionable feedback for maintaining model integrity as systems evolve over time [10], [3].

Blockchain technology introduces a decentralized paradigm for establishing immutable, transparent traceability records. In their study, [6] presented a dual-level traceability framework that records artifact history and requirement evolution within secure, distributed ledgers. This approach addresses trust and collaboration issues endemic to multi-stakeholder and geographically dispersed projects while supporting enhanced auditability and resistance to tampering. The experiential study underscores blockchain's suitability for critical system engineering projects where provenance and consensus are paramount.

Other specialized modeling frameworks include ontology-based systems [17], which codify domain semantics to enable advanced reasoning and recommendation functions, and NLP-driven techniques [13], [12], which automate the extraction of traceability links by parsing large corpora of requirements, code, and documentation. These approaches illustrate the trend toward leveraging semantic, linguistic, and knowledge-based structures as integral modeling elements for traceability.

Table 1 *Traceability modeling techniques and tools*

Reference	Technique Type	Description/Approach	Tools/Framework used
[4]	ML-based Formal Method	GA-XWCoDe (XGBoost + Node2Vec + GA for R2C traceability)	Custom GA-XWCoDe
[16]	Deep Learning	MS-CodeBERT fine-tuned for multi-type traceability prediction	MS-CodeBERT
[3]	Graph + Traces	BPMN + execution trace for enhanced link recovery	METRA
[6]	Blockchain	Dual-level immutable traceability framework	Blockchain ledger
[10]	Architecture Mapping	Seamless system architecture model via static analysis	Not specified
[17]	Ontology-Based	Mapping connectors with execution views	Ontology rules
[13]	ML Performance Model	Traceability via test-based performance learning	METRA
[12]	NLP-Based	Systematic framework for IoT system spec creation	ChatGPT
[21]	NLP + Deep Learning	MS-CodeBERT with fine-tuned training for traceability prediction	MS-CodeBERT
[9]	Semantic Similarity	Linking matrix generation using semantic NLP	Custom semantic matrix tool
[15]	NLP Systematic Review	NLP role in traceability (review study)	Review synthesis
[8]	Tool Integration	GitHub Marketplace survey on traceability and automation	GitHub Marketplace tools

3.1 Tools Integration Approaches

Effective traceability relies not only on sophisticated models but also on seamless integration with development environments and workflows. Modern practice increasingly emphasizes embedding traceability tools within DevOps pipelines and leveraging marketplace platforms for automation.

The GitHub Marketplace epitomizes this shift, providing a rich ecosystem for production automation and traceability tool integration. As demonstrated by [8], practitioners tend to favor tools that support continuous integration, utilities, and code quality monitoring. Their findings highlight a divergence between academic tool development and industrial adoption, signaling the importance of responsive tool curation and robust integration paths.

AI-assisted traceability augmentation has also gained prominence. While large language models (LLMs) such as ChatGPT have proven adept at specification and requirements drafting [14], their application to traceability remains nuanced. Tools like DejaVu enhance developers' understanding of system evolution by integrating code changes, commit history, and issue-tracking data into visualization dashboards [18]. In their work, [19] explored how generative AI, coupled with prompt engineering, can automate property verification and requirement change activities, showcasing the potential for AI to expedite traceability processes while supporting human-in-the-loop collaboration.

Automated tracing is further streamlined with machine-learning classifiers that mine commit histories and recommend refactoring actions [20] or classify issue reports based on user-manual features using deep-learning

embeddings [21]. These techniques are often tightly coupled with existing configuration-management and CI/CD infrastructures, enhancing traceability precision without disrupting developer workflows.

Ontology-driven recommendation systems [17] and semantic-similarity-based requirement-linking platforms [9] have also been deployed to bridge the gap between requirements and diverse software artifacts. Meanwhile, advanced traceability solutions can reconstruct execution architectures and dependency views directly from source code, further aiding traceability and system comprehension [22].

4. Result and Discussions

This section presents the findings of the review, based on 24 recent studies on software traceability published between 2022 and 2025. The papers were categorized according to their primary focus areas: performance of modeling-based approaches, tool integration and automation, practical challenges, and future research directions. Figure 2 and Figure 3 visualize the comparative performance of traceability techniques and their chronological development, respectively.

4.1 Performance of Modeling-Based Approaches

Recent advancements in software traceability have introduced a variety of modeling techniques, each addressing accuracy, scalability, and maintainability from different perspectives. As shown in Figure 1, the heatmap of traceability techniques versus evaluation criteria compares popular approaches such as GA-XWCoDe, MS-CodeBERT, Blockchain, DAQUAVORD, and Ontology-based recommendations across five dimensions: accuracy, scalability, usability, automation, and trustworthiness.

GA-XWCoDe [4], which integrates XGBoost, Node2Vec, and genetic algorithms, achieved a 17.44–33.36 % improvement in F1-scores compared with earlier models. Similarly, MS-CodeBERT [16], employing code data augmentation and fine-tuning, demonstrated precision gains of up to 8.5 %. Both techniques emphasize the growing role of machine-learning formalism in enhancing link recovery accuracy and scalability.

To better contrast their capabilities, Table 2 presents a side-by-side comparison of GA-XWCoDe and MS-CodeBERT in terms of input data type, learning approach, evaluation metrics, computational requirements, and practical applicability. This comparative overview helps clarify not only their strengths, but also the scenarios in which each technique might be better suited.

Table 2 *Comparative performance of modeling-based traceability techniques*

Criteria	GA-XWCoDe (ML Hybrid)	MS-CodeBERT (Deep Learning)
Technique Components	Genetic Algorithm + XGBoost + Node2Vec	Code augmentation + Pretrained Transformer
Data Type	Requirements-to-code (multi-language)	Java code and issue descriptions
Performance (F1-score)	17.44%–33.36% improvement over baselines	Up to 7% F1-score gain after fine-tuning
Precision	High; especially effective on noisy inputs	Improved by 8.5% with data augmentation
Scalability	Robust on varying project sizes	Suitable for large-scale Java repositories
Strengths	Multi-model integration; explainable process	Pretrained efficiency; adaptable to variants
Limitations	Complex tuning; may overfit hybrid weights	Domain-limited; tuning needed for transfer
Best Use Case	Complex multi-language enterprise systems	Java-based OSS or issue tracking automation

In contrast, Blockchain-based traceability [6] performs strongly in trustworthiness but has lower usability, indicating trade-offs between security and accessibility. Ontology-driven and NLP-based models [12], [19] provide balanced yet moderate outcomes, supporting explainability and semantic precision without heavy computational overhead. Figure 2 shows a heatmap of traceability technique with evaluation criteria.

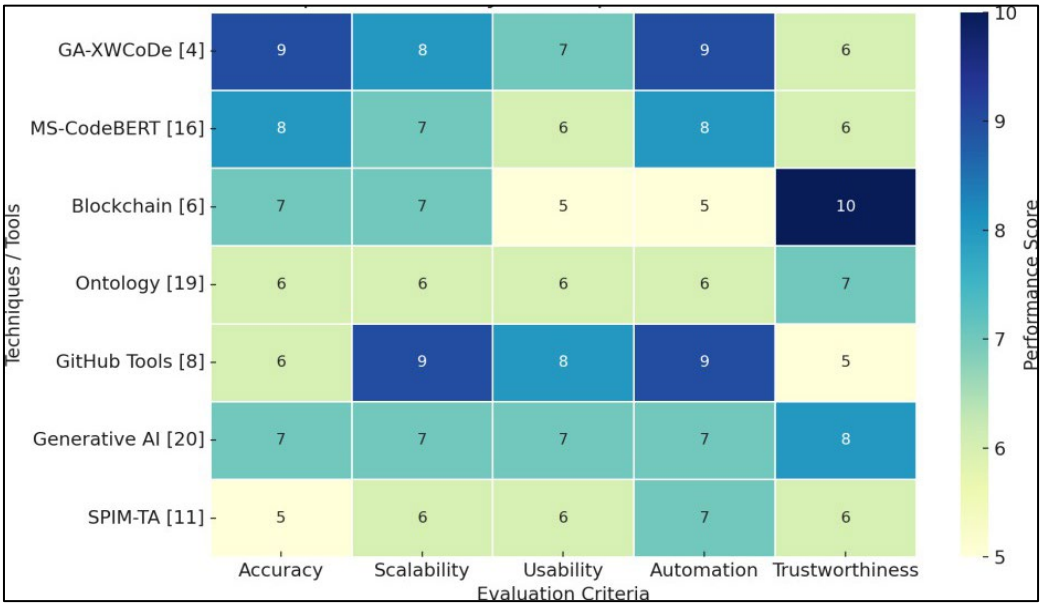

Fig. 2 *Heatmap of traceability techniques vs evaluation criteria*

Figure 2 illustrates the relationship between traceability techniques and evaluation criteria. This heatmap compares traceability techniques such as GA-XWCoDe, MS-CodeBERT, Blockchain, DAQUAVORD, Ontology-based Recommendation, and others across five key evaluation dimensions: Accuracy, Scalability, Usability, Automation, and Trustworthiness. Higher intensity (darker color) indicates stronger performance under that criterion. This figure highlights how different methods emphasize different strengths. For example, GA-XWCoDe and MS-CodeBERT excel in Accuracy and Scalability, while Blockchain leads in Trustworthiness but lags in Usability. Ontology-based approaches are balanced but moderate in most criteria. The heatmap provides a visual guide to inform method or tool selection in alignment with project needs.

In the context of tool integration, as examined by [8], the GitHub Marketplace highlights the practical value of automation tools for production workflows such as Continuous Integration and utility support. Although automation topics have high traction in both academia and industry, their study reveals a misalignment between research emphasis (more on code quality and testing) and industry usage (more on production automation), reflecting an ongoing need for tool usability that bridges research and operational necessity. The authors also underscore the challenges of aligning tool features with diverse process requirements in real-world DevOps pipelines.

As reported in [22], the usability of execution-architecture reconstruction is evaluated through a systematic, domain-agnostic method of mapping source-code connectors. Their empirical results show soundness and efficiency, reconstructing execution architectures with over an 86 percent F1-score using less than 13.9 person-hours, underscoring both practical applicability and the method's scalability for large, unstructured systems.

Other notable models, such as METRA proposed by [3], expand BPMN models using execution traces and further improve traceability-link recovery by enhancing linguistic cues while maintaining high precision (over 74 percent) and recall. Meanwhile, research by [6] introduced blockchain's capability in traceability, ensuring secure, immutable trace logs that are particularly beneficial in distributed-stakeholder environments. However, these solutions may introduce complexity and infrastructural overheads not always suited for all organizational scales or legacy contexts.

4.2 Tool Integration and Automation

Beyond algorithmic innovations, the integration of traceability techniques into developer workflows has become increasingly important. Tool ecosystems such as GitHub Marketplace and DevOps pipelines play a central role in automating traceability, reducing manual workload, and enhancing project scalability.

In their large-scale analysis of over 2,000 tools on GitHub Marketplace, [8] highlighted that a majority of popular traceability-related plugins were aligned with Continuous Integration (CI), build automation, and release workflows. Despite the prominence of research on code-quality and test generation, many widely adopted tools instead focus on deployment readiness, ticket tracking, and runtime monitoring. This misalignment reveals a practical gap: what researchers optimize for (code-level accuracy) differs from what practitioners prioritize (deployment automation and traceability in production environments).

DejaVu, a traceability-supporting plugin designed for GitHub Actions, offers contextual linking between commit messages, issues, and test outcomes. It exemplifies how automation can be embedded into existing DevOps pipelines, supporting developers with near real-time feedback and reduced documentation overhead. However, tools like DejaVu also highlight ongoing challenges that most notably, limited configurability, lack of explainable trace recommendations, and difficulty scaling across large or modular repositories.

Meanwhile, traceability tooling integrated into CI/CD workflows demonstrates strong automation potential, but still faces barriers in terms of cross-tool interoperability and consistent metadata standards. For instance, [22] noted that even with sound architectural reconstruction models, integrating trace links into version-control logs or runtime dashboards requires additional engineering effort and stakeholder cooperation.

Together, these findings suggest that while automated traceability is gaining momentum in the tool landscape, deeper alignment between academic models and engineering platforms is needed. Future efforts should prioritize usability, extensibility, and seamless DevOps integration as core evaluation criteria.

4.3 Practical Challenges

Together, these findings suggest that while automated traceability is gaining momentum in the tool landscape, deeper alignment between academic models and engineering platforms is needed. Future efforts should prioritize usability, extensibility, and seamless DevOps integration as core evaluation criteria.

While modeling-based techniques and integrated tooling mark significant advancements in software traceability, their translation into real-world practice remains limited due to several persistent challenges that many of which are evident even in the top-performing models and tools reviewed in this study.

First, standardization remains elusive. Techniques like GA-XWCoDe [4] and MS-CodeBERT [16], despite their high accuracy and scalability, depend on specific input structures and fine-tuned pipelines, making them difficult to generalize across diverse projects or organizations. Similarly, ontology-based approaches [19] offer semantic clarity but lack universal schemas or alignment standards, limiting their interoperability across domains or tools.

Second, stakeholder collaboration and engagement remain labor-intensive, particularly during traceability link validation. The DAQUAVORD framework [5], though methodical, encountered difficulties reconciling divergent stakeholder views during traceability mapping. This issue is echoed in [7], where iterative refinements improve technical scalability, yet consensus-building across heterogeneous teams remains a bottleneck. Even when models perform well, such as MS-CodeBERT's 8.5% precision gain, their adoption depends heavily on developers' trust and understanding something not easily addressed through algorithmic improvements alone.

Third, tool integration often introduces new usability and process challenges. While automation tools in GitHub Marketplace [8] and DejaVu streamline traceability workflows, their configurations are often rigid, poorly documented, or poorly aligned with evolving DevOps practices. As [22] demonstrated, even high-performing architectural reconstruction tools (86% F1-score) require significant manual setup effort, pointing to the fragility of tool usability in production environments.

Lastly, reliance on AI-powered solutions introduces transparency and verification risks. ChatGPT and similar tools [14], [19], while promising in automating trace link suggestions or requirement synthesis, still struggle with explainability, verification against gold standards, and resolving conflicts in stakeholder inputs. These concerns highlight that accuracy alone is insufficient; practical traceability requires tools and techniques that are explainable, adaptable, and support collaborative human oversight.

In summary, the reviewed studies emphasize that future traceability research must look beyond algorithmic improvements to address contextual, human-centered, and organizational constraints that currently hinder scalable adoption.

4.4 Future Directions

Looking ahead, integrating artificial intelligence (AI) and blockchain technologies offers promising research directions to address some of the aforementioned challenges. As advocated by [23], broader adoption of AI-assisted tools can support sustainability-driven requirements management. Building on this, studies by [19] and [24] propose roadmaps that leverage large language models and formal methods to ensure correctness, fairness, and trustworthiness in requirements engineering, thereby supporting more explainable and accessible traceability mechanisms. Figure 3 shows the timeline of major traceability innovations.

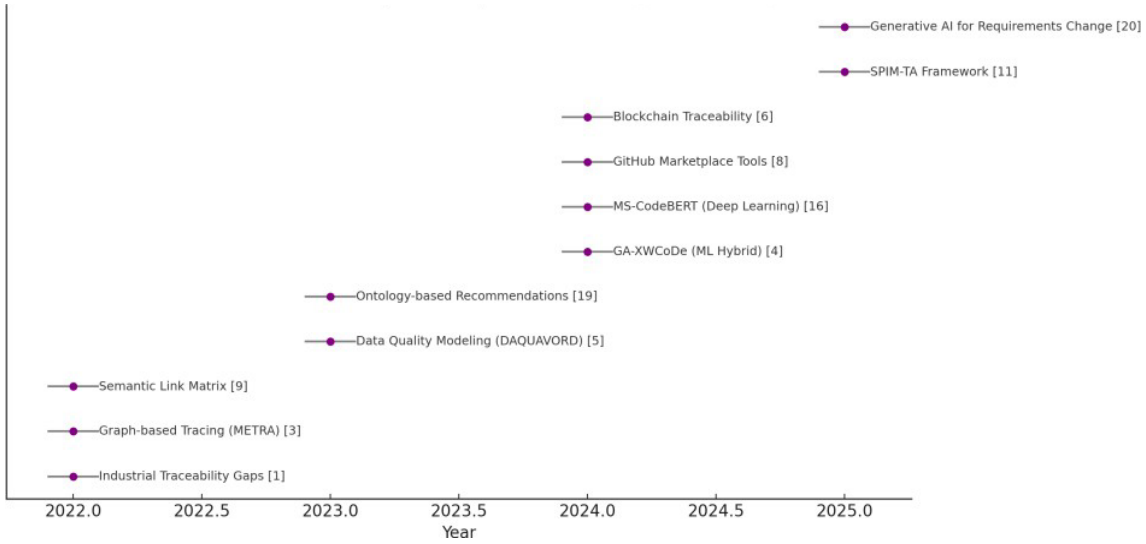

Fig. 3 *Timeline of major traceability innovations from 2022 to 2025*

Figure 3 shows the progression of major innovations in software traceability between 2022 and 2025. Early contributions in 2022 include studies addressing industrial traceability gaps [1], the graph-based METRA model [3], and the Semantic Link Matrix [9]. In 2023, progress was marked by Data Quality Modeling (DAQUAVORD) [5] and Ontology-based Recommendations [19]. The year 2024 witnessed rapid growth with the introduction of GA-XWCoDe [4], MS-CodeBERT [16], GitHub Marketplace integration [8], and Blockchain Traceability [6]. Recent frameworks in 2025 include the SPIM-TA maturity model [11] and Generative AI for managing Requirements Change [20]. This timeline reflects the accelerating pace of innovation in modeling techniques, tool integration, and automation for software traceability.

As outlined by [6], blockchain-based traceability systems present secure, auditable solutions particularly suitable for multi-stakeholder and regulation-driven development environments. These systems can support dual-level traceability and provide immutable artifact histories, fostering greater stakeholder trust and transparency.

Holistic frameworks and standardization efforts are needed to better support interoperability and process integration. As recommended by [12], comprehensive NLP-based frameworks should address representation similarity, model scalability, and explainability, which are key aspects for future traceability solutions to achieve widespread and adaptable adoption. Similarly, the convergence of model-driven, automated-test-based, and AI-enabled approaches can yield on-demand, context-aware traceability support, which is crucial for managing increasingly complex and variant-rich software ecosystems.

5. Conclusion

This review has synthesized recent advancements in software traceability by analyzing 24 peer-reviewed studies published between 2022 and 2025. The study contributes by categorizing these works into four key themes: performance of modeling-based approaches, tool integration and automation, practical challenges, and future research directions. Through comparative analysis, this paper highlights the strengths of emerging techniques such as GA-XWCoDe and MS-CodeBERT, while also offering visual tools. It includes a concept map, timeline, and heatmap to support researchers and practitioners in understanding the evolving landscape of traceability solutions.

Despite notable progress, several gaps remain. Challenges persist in terms of standardization, stakeholder collaboration, explainability, and the real-world applicability of traceability frameworks. In particular, the lack of unified evaluation benchmarks and insufficient empirical validation across diverse development environments continue to hinder large-scale adoption.

Future research must address these unresolved issues by developing scalable, explainable, and domain-adaptable traceability models. There is also a pressing need to align academic innovations with industry practices through open-source tooling, AI integration, and collaborative validation across stakeholder ecosystems. Bridging these gaps will be crucial for realizing robust, sustainable, and context-aware traceability solutions in modern software engineering.

Acknowledgement

We would like to express our appreciation for the support provided by all parties directly or indirectly involved in this research. We also thank our colleagues and collaborators for their support and assistance throughout the research process.

Conflict of Interest

The authors declare that there is no conflict of interest regarding the publication of the paper.

Author Contribution

The authors are responsible for the study conception, research design, data collection, data analysis, result interpretation and manuscript drafting.

References

[1] Fucci, D., Alégroth, E., & Axelsson, T. (2022). When traceability goes awry: An industrial experience report. *Journal of Systems and Software, 192*, 111389. https://doi.org/10.1016/j.jss.2022.111389

[2] Włodarski, R., Poniszewska-Marańda, A., & Falleri, J.-R. (2022). Impact of software development processes on the outcomes of student computing projects: A tale of two universities. *Information and Software Technology, 144*, 106787. https://doi.org/10.1016/j.infsof.2021.106787

[3] Lapeña, R., Pérez, F., Pastor, Ó., & Cetina, C. (2022). Leveraging execution traces to enhance traceability links recovery in BPMN models. *Information and Software Technology, 146*, 106873. https://doi.org/10.1016/j.infsof.2022.106873

[4] Zou, Z., Wang, B., Hu, X., Deng, Y., Wan, H., & Jin, H. (2024). Enhancing requirements-to-code traceability with GA-XWCoDe: Integrating XGBoost, Node2Vec, and genetic algorithms. *Journal of King Saud University – Computer and Information Sciences, 36*, 102197. https://doi.org/10.1016/j.jksuci.2024.102197

[5] Guerra-García, C., Nikiforova, A., Jiménez, S., Perez-Gonzalez, H. G., & Ramírez-Torres, M. (2023). ISO/IEC 25012-based methodology for managing data quality requirements. *Data and Knowledge Engineering, 145*, 102152. https://doi.org/10.1016/j.datak.2023.102152

[6] Abrar, M. F., Alharbi, Y., Alsaffar, M., Hussain, S., Saqib, M., Khan, J., & Lee, Y. (2024). Blockchain technology for requirement traceability in systems engineering. *Information Systems, 123*, 102384. https://doi.org/10.1016/j.is.2024.102384

[7] Sadovykh, A., Said, B., Truscan, D., & Bruneliere, H. (2024). An iterative approach for model-based requirements engineering in large collaborative projects. *Science of Computer Programming, 232*, 103047. https://doi.org/10.1016/j.scico.2024.103047

[8] Golam Saroar, S. K., Ahmed, W., Onagh, E., & Nayebi, M. (2024). GitHub marketplace for automation and innovation in software production. *Information and Software Technology, 175*, 107522. https://doi.org/10.1016/j.infsof.2024.107522

[9] Rajpathak, D., Peranandam, P. M., & Ramesh, S. (2022). Automatic development of requirement linking matrix based on semantic similarity. *Journal of Systems and Software, 186*, 111211. https://doi.org/10.1016/j.jss.2021.111211

[10] Tsiperman, G. (2025). Design techniques for a seamless information system architecture. *Procedia Computer Science, 256*, 308–318. https://doi.org/10.1016/j.procs.2025.02.031

[11] Abrar, M. F., Alharbi, Y., Alsaffar, M., Hussain, S., Saqib, M., Khan, J., & Lee, Y. (2025). Developing SPIM-TA: A maturity-level framework for software testing automation. *Ain Shams Engineering Journal, 16*, 103472. https://doi.org/10.1016/j.asej.2025.103472

[12] Binder, M., & Mezhuyev, V. (2024). A framework for creating an IoT system specification with ChatGPT. *Internet of Things, 27*, 101218. https://doi.org/10.1016/j.iot.2024.101218

[13] Camilli, M., Janes, A., & Russo, B. (2022). Automated test-based learning and verification of performance models for microservices systems. *Journal of Systems and Software, 187*, 111225. https://doi.org/10.1016/j.jss.2022.111225

[14] Gupta, A., Poels, G., & Bera, P. (2023). Generating multiple conceptual models from behavior-driven development scenarios. *Data and Knowledge Engineering, 145*, 102141. https://doi.org/10.1016/j.datak.2023.102141

[15] Pauzi, Z., & Capiluppi, A. (2023). Applications of natural language processing in software traceability. *Journal of Systems and Software, 198*, 111616. https://doi.org/10.1016/j.jss.2023.111616

[16] Wang, B., Zou, Z., Wan, H., Li, Y., Deng, Y., & Li, X. (2024). An empirical study on requirement-to-code traceability link recovery. *Journal of King Saud University – Computer and Information Sciences, 36*, 102118. https://doi.org/10.1016/j.jksuci.2024.102118

[17] Ahn, H., Kang, S., & Lee, S. (2025). Reconstruction of an execution architecture view by identifying mapping rules for connectors. *Journal of Systems and Software, 220*, 112268. https://doi.org/10.1016/j.jss.2025.112268

[18] Cho, H., Lee, S., & Kang, S. (2022). Classifying issue reports based on feature descriptions using deep learning. *Information and Software Technology, 142*, 106743. https://doi.org/10.1016/j.infsof.2021.106743

[19] El Alaoui, M., Chapurlat, V., Rabah, S., Richet, V., & Plana, R. (2023). An ontology-based approach for recommendation on systems engineering projects. *Procedia Computer Science, 225*, 1350–1359. https://doi.org/10.1016/j.procs.2023.03.162

[20] Kong, Y., Zhang, N., Duan, Z., & Yu, B. (2025). Collaboration with generative AI to improve requirements change. *Computer Standards & Interfaces, 94*, 104013. https://doi.org/10.1016/j.csi.2025.104013

[21] Majidzadeh, A., Ashtiani, M., & Zakeri-Nasrabadi, M. (2024). Multi-type requirements traceability prediction with MS-CodeBERT. *Computer Standards & Interfaces, 90*, 103850. https://doi.org/10.1016/j.csi.2023.103850

[22] Nyamawe, A. S. (2022). Mining commit messages to enhance software refactorings recommendation. *Machine Learning with Applications, 9*, 100316. https://doi.org/10.1016/j.mlwa.2022.100316

[23] Rahimi, M., & Vierhauser, M. (2022). Visualization of aggregated information to support software evolution. *Journal of Systems and Software, 192*, 111421. https://doi.org/10.1016/j.jss.2022.111421

[24] Watz, M., & Hallstedt, S. I. (2022). Towards sustainable product development: A profile model for design requirements. *Journal of Cleaner Production, 346*, 131000. https://doi.org/10.1016/j.jclepro.2022.131000