Using NLP-Based Traceability Recovery to Reduce Architectural Complexity from Requirement Volatility in Software Systems

OPEN ACCESS

Volume: 13

Special Issue: 2

Month: January

Year: 2026

E-ISSN: 2582-0397

P-ISSN: 2321-788X

Citation:
Adu, Edward Obeng, et al. "Using NLP-Based Traceability Recovery to Reduce Architectural Complexity from Requirement Volatility in Software Systems." *Shanlax International Journal of Arts, Science and Humanities*, vol. 13, no. 2, 2026, pp. 126–31.

DOI:
https://doi.org/10.34293/sijash.v13iS2-i1-Jan.10469

Edward Obeng Adu
Master Student, Department of Computing and Engineering
Heritage Christian University, Accra, Greater Accra, Ghana

Mary Immaculate Sheela L
Faculty, Department of Computing and Engineering
Heritage Christian University, Accra, Greater Accra, Ghana

Dr. Sasikala P.
Faculty, Department of Computer Science
Lal Bahadur Shastri Government First Grade College, Bengaluru, Karnataka, India

Abstract
This study addresses the problem of requirements–architecture traceability deterioration in software-intensive systems. Frequent changes in requirements weaken architectural documentation and design rationale, reducing visibility into change impact and contributing to architectural drift and increased system complexity. Leveraging advances in NLP-based traceability link recovery, this research adopts a design science methodology to design and evaluate an automated traceability recovery system. The proposed system reconstructs missing links between requirements and architectural artifacts, including Software Architecture Document sections, Architecture Decision Records, and component descriptions. It integrates transformer-based semantic models such as BERT and RoBERTa, similarity search using FAISS, and ontology-supported reasoning through a Neo4j knowledge graph. These components enable cross-artifact traceability and support explainable validation with human involvement. Artifacts obtained from open-source repositories and expert-validated requirements are used to establish a reference dataset. Trace link accuracy is evaluated using precision, recall, and F1-score, along with task-based evaluation of change impact analysis under volatile requirement conditions. The results are expected to demonstrate that automated traceability recovery improves change impact understanding and helps control architectural complexity during system evolution.

Keywords: Natural Language Processing (NLP), Large Language Models (LLM), Traceability Link Recovery (TRL), Artifacts, Software Architecture Document (SAD), Architecture Decision Record (ADR)

Introduction

Requirement volatility is common in software-intensive systems and affects architectural stability by requiring frequent updates to design decisions, documentation, and implementation (Mohammad & Kollamana 2024). Each requirement change can propagate across design artifacts, source code, and test cases. When trace links are missing or outdated, developers may not fully understand change impact, which contributes to

increased architectural complexity (Yaşa et al., 2025). Inadequate traceability between requirements and the actual artifact causes this problem because the artifact lacks accurate links that reveal which component implements the requirement and how the changes circulate through the system. According to Guo et al. (2025), requirement traceability means being able to link each requirement to its sources, such as stakeholder rationales, hazards, and regulations, and forward to design artifacts, code elements, and test cases, hence supporting verification and validation, change impact analysis, compliance assessments, safety analysis, and regression testing. However, maintaining these trace links manually requires significant effort and is prone to human error, reducing its effectiveness in large and evolving software projects (Ahmadiyah et al., 2023). Recent advances in natural language processing provide effective mechanisms for automating requirements traceability (Necula et al., 2024).

NLP-based traceability link recovery (TLR) provides a practical mechanism to reconstruct missing requirement–architecture links from natural language artifacts such as Software Requirement Specifications, Software Architecture Documents, issue reports, and Architecture Decision Records. NLP methods, from information retrieval to embeddings and LLM–assisted approaches, are systematic evidence that trace links with measurable accuracy, and less manual effort can be achieved with NLP. Emerging work demonstrates the feasibility of LLM–based architecture traceability by isolating architectural concepts and aligning them with documentation sources.

Literature Survey

Mosquera et al. (2025) classified existing research on automated software trace link discovery into two main categories: artifact-centric approaches and knowledge-representation-based approaches. Artifact-centric approaches typically rely on project-specific datasets, training data, or parsing mechanisms to identify trace links between software artifacts. However, these approaches may face limitations when applied across different projects, particularly when labeled datasets are unavailable. In contrast, knowledge-representation-based approaches use structured representations such as ontologies, taxonomies, or metamodels to enhance semantic understanding and enable traceability across heterogeneous artifacts. These approaches allow artifacts to be translated into a unified representation and support trace link identification through configurable semantic relationships. Despite these advantages, many tools still require human validation to confirm trace link accuracy. Furthermore, several studies primarily evaluate traceability techniques based on accuracy metrics, with limited focus on usability and practical effectiveness in real development environments.

Mosquera et al. (2025) further emphasized the need for comprehensive evaluation of hybrid traceability tools such as OntoTrace V2.0, which combines ontology-based reasoning with NLP similarity analysis. Such tools have the potential to improve trace link discovery by leveraging both semantic knowledge and textual similarity. However, effective evaluation should not be limited to traditional performance metrics such as precision and recall. It is also important to examine usability, integration feasibility, and the extent to which such tools support developers in real software engineering workflows.

Lan et al. (2023) highlighted the importance of traceability between diverse software artifacts, particularly issue reports and commit histories, which are essential for software maintenance tasks such as bug localization and defect prediction. In many development environments, developers manually include issue identifiers in commit messages to establish traceability links. However, this process is often inconsistent, resulting in missing or incomplete links. Traditional automated approaches that rely on rule-based methods have shown limited effectiveness because they cannot capture complex semantic relationships present in natural language and source code.

To address this limitation, Lan et al. (2023) proposed BTLink, a traceability recovery approach based on pretrained BERT models. The approach encodes issue descriptions and commit messages into semantic representations and uses these representations to determine trace link relationships. Their experimental evaluation across multiple open-source projects demonstrated improved performance compared to existing

methods. The results indicated that transformer-based semantic models can effectively capture contextual relationships and improve traceability link recovery accuracy across different software projects.

Necula et al. (2024) conducted a comprehensive review of NLP applications in software requirements engineering, covering research published between 1991 and 2023 across major scientific databases such as Scopus, IEEE Xplore, ACM Digital Library, and Clarivate. Their review identified two primary research directions: automation of requirement engineering tasks and improvement of requirement quality. NLP techniques have been applied to automate tasks such as requirement extraction, classification, and model generation, as well as to detect ambiguity and inconsistencies in requirement specifications. Machine learning, deep learning, and large language models have played an increasingly important role in enabling these capabilities.

The review also identified additional research areas, including semantic analysis, requirement clustering, and traceability recovery. NLP-based tools have demonstrated the potential to improve requirement engineering processes and enhance traceability. However, several challenges remain, including handling domain-specific terminology, ensuring reliability, and integrating NLP techniques into practical software development workflows (Siddique, 2024). These challenges highlight the need for more robust, adaptable, and context-aware traceability recovery approaches. Future research should focus on improving semantic understanding, system integration, and practical usability to ensure effective traceability management in real-world software engineering environments.

Research Gaps

Existing studies demonstrate significant progress in applying natural language processing techniques to automate requirements engineering and traceability recovery. Several approaches have shown success in recovering specific trace links, such as issue–commit relationships, using semantic analysis and machine learning methods. However, limited empirical evidence exists to confirm whether recovered trace links lead to measurable improvements in practical software engineering outcomes, such as enhanced change impact analysis, reduced architectural rework, or controlled architectural complexity under conditions of requirement volatility. Most existing research evaluates performance primarily using dataset-level accuracy metrics, including precision, recall, and F1-score, without sufficiently examining the practical benefits of traceability recovery in real development environments.

Another important limitation is the fragmented focus of existing approaches across different artifact types. Many traceability recovery methods address relationships between requirements and source code or between issues and commits, but fewer studies consider traceability between requirements and architectural artifacts such as architecture decision records and architecture documentation. This lack of comprehensive artifact integration restricts the effectiveness of traceability recovery in managing architectural evolution and complexity.

In addition, challenges related to contextual understanding, domain-specific terminology, and cross-project applicability continue to affect the reliability of automated traceability methods. Many approaches also lack mechanisms to incorporate human validation and feedback, which are essential for ensuring trace link accuracy and practical usability. Furthermore, traceability recovery is often treated as a one-time task rather than as a continuous process that supports ongoing system evolution. These limitations highlight the need for a traceability recovery approach that integrates multiple artifact types, supports human-in-the-loop validation, and evaluates effectiveness based on real-world impact analysis and architectural management outcomes. Addressing these gaps can contribute to improved traceability maintenance and more effective control of architectural complexity in software systems affected by requirement volatility.

Problem Statement / Formulation

Requirement volatility significantly affects the stability and maintainability of software architecture. Frequent changes in requirements weaken the consistency of architectural documentation and gradually

increase architectural complexity, making software systems more difficult to evolve in a controlled and reliable manner (Siakas et al., 2022). In many software projects, traceability links between requirements and architectural elements are incomplete from the early stages of development and continue to degrade as the system evolves. Maintaining these links manually requires considerable effort and is prone to human error, especially in large and dynamic software environments.

When traceability information is missing or outdated, developers lack clear visibility into how requirement changes influence architectural components and design decisions. This limited visibility often results in reactive and localized architectural modifications rather than systematic updates based on complete system understanding. Over time, such practices contribute to architectural drift, increased system complexity, and reduced maintainability (Cervantes & Kazman, 2024). Without reliable traceability support, managing requirement changes becomes more difficult, and the risk of introducing architectural inconsistencies increases.

Therefore, there is a need for an automated traceability recovery approach that can reconstruct missing links between requirements and architectural artifacts. Such an approach can improve change impact analysis, support informed architectural decision-making, and help control architectural complexity in software systems affected by requirement volatility.

Objectives

The primary aim of this research is to develop and evaluate an NLP-based traceability recovery approach that reconstructs trace links between requirements and architectural artifacts. The proposed approach aims to improve change impact analysis and reduce architectural complexity caused by requirement volatility.

To achieve this aim, the study focuses on the following objectives:

- To evaluate the accuracy of recovered trace links using standard metrics such as precision, recall, and F1-score against a validated ground truth.
- To design an NLP-based method capable of automatically recovering traceability links between requirements and architectural artifacts, including Software Architecture Documents, Architecture Decision Records, and component descriptions.
- To assess the practical impact of recovered traceability links on system evolution by analyzing change impact outcomes, including analysis effort, missed impacts, and incorrect impact identification.

Methodology

This study adopts a design science methodology combined with a mixed-method approach to develop and evaluate an NLP-based traceability recovery system. The design science approach emphasizes the development and rigorous evaluation of a practical solution artifact, namely the proposed traceability recovery system, rather than focusing only on problem analysis (Tuunanen et al., 2024). The approach addresses requirement volatility in software-intensive systems by automatically reconstructing trace links between requirements and architectural artifacts. This enables more effective change impact analysis and helps control the growth of architectural complexity during system evolution.

The primary unit of analysis in this study is a traceability link between a requirement and its corresponding architectural artifact, such as Software Architecture Document sections, Architecture Decision Records, or component descriptions. Secondary units include requirement change scenarios, downstream impact analysis outcomes such as missed or incorrect impact identification, and architectural complexity indicators such as propagation scope and component dependency concentration. Where available, additional artifacts including issue reports, commit histories, and source code components are incorporated to provide richer context and support more accurate traceability recovery.

Software artifacts are collected from open-source projects that provide accessible documentation, version histories, and architectural descriptions. The dataset includes requirement specifications, architecture

documentation, architecture decision records, component descriptions, source code, and commit logs. The system extracts textual and structured data from these artifacts and performs preprocessing steps such as cleaning, segmentation, and normalization to prepare them for analysis. A validated reference set of requirement–architecture trace links is established through expert annotation and verification, which serves as ground truth for evaluating system performance.

The system development process consists of multiple phases. The first phase involves collecting artifacts and preprocessing the data to define traceability units and requirement change scenarios. The second phase focuses on knowledge representation by constructing a Neo4j knowledge graph that models relationships between requirements and architectural elements. An ontology is defined to capture semantic relationships such as "implements," "depends on," and "affects," which enables structured reasoning across artifacts. The third phase implements the traceability recovery engine using transformer-based semantic models and embedding similarity techniques. Vector similarity search is applied to generate candidate trace links, which are further refined using ontology-based reasoning and validation support. The final phase integrates all system components and performs iterative refinement to improve trace link accuracy, explainability, and system reliability as requirements evolve.

The implementation uses Python for artifact extraction, preprocessing, and semantic analysis. Transformer-based models such as BERT and RoBERTa are used to generate semantic embeddings that capture contextual relationships between artifacts. Neo4j is used to store and manage the knowledge graph, and FAISS is used to enable efficient similarity search and retrieval of candidate trace links. Git and issue-tracking APIs are used to extract project history and artifact relationships. The system is deployed on a workstation equipped with an 8–16 core CPU, 32 GB RAM, SSD storage, and a CUDA-enabled GPU to support efficient semantic processing and model inference.

Expected Outcomes

This research is expected to produce a functional prototype that applies natural language processing techniques to recover traceability links between software requirements and architectural artifacts. The proposed system will assist developers in identifying, verifying, and maintaining trace links more effectively than manual methods. By reconstructing missing or incomplete links, the system will improve the accuracy of change impact analysis and help developers understand how requirement changes affect architectural components. The approach is also expected to demonstrate improved trace link recovery performance when compared to existing methods, particularly in environments with frequent requirement changes. In addition, the recovered trace links will support better architectural decision-making, reduce the likelihood of overlooked dependencies, and help control architectural complexity as software systems evolve over time.

Conclusions and Future Scope

This research demonstrates the potential of natural language processing and transformer-based semantic analysis to recover traceability links between requirements and architectural artifacts in software-intensive systems. The proposed approach improves traceability completeness and enhances the ability to analyze the impact of requirement changes on system architecture. By integrating semantic similarity analysis, knowledge graph representation, and ontology-based reasoning, the system provides a structured and scalable method for maintaining traceability during software evolution. Improved traceability enables developers to make informed architectural decisions, reduces the risk of architectural drift, and supports long-term system maintainability.

Future work will focus on extending the proposed system to support more complex traceability relationships across requirements, architecture, source code, test cases, and runtime artifacts. Incorporating continuous learning from developer feedback can further improve trace link accuracy and adapt the system to project-specific terminology and practices. Additional evaluation using large-scale industrial software projects will

help validate system effectiveness in real-world environments. Integration with software development tools can enable real-time traceability support and facilitate continuous architectural monitoring. Further research may also explore improving system scalability, explainability, and robustness to ensure reliable traceability recovery in diverse and evolving software systems.

References

1. Ahmadiyah, A., Rochimah, S., & Siahaan, D. (2023). Monthes: A compact software traceability dataset (pp. 55–61).
2. Cervantes, H., & Kazman, R. (2024). Designing Software Architectures: A Practical Approach. Addison-Wesley Professional.
3. Guo, J. L. C., Steghöfer, J.-P., Vogelsang, A., & Cleland-Huang, J. (2025). Natural language processing for requirements traceability. In Handbook on Natural Language Processing for Requirements Engineering (pp. 89–116). Springer Nature Switzerland.
4. Lan, J., Gong, L., Zhang, J., & Zhang, H. (2023). BTLink: Automatic link recovery between issues and commits based on pre-trained BERT model. *Empirical Software Engineering, 28*(4).
5. Mohammad, A., & Kollamana, J. M. (2024). Causes and mitigation practices of requirement volatility in agile software development. *Informatics (MDPI), 11*(1), 12. doi: 10.3390/informatics11010012.
6. Mosquera, D., Ruiz, M., & Pastor, O. (2025). Ontology-based NLP tool for tracing software requirements and conceptual models: An empirical study. *Requirements Engineering, 30*(3), 341–369. doi: 10.1007/s00766-025-00447-4.
7. Necula, S.-C., Dumitriu, F., & Greavu-Șerban, V. (2024). A systematic literature review on using natural language processing in software requirements engineering. *Electronics, 13*(11), 2055. doi: 10.3390/electronics13112055.
8. Siakas, E., Rahanu, H., Georgiadou, E., & Siakas, K. (2022). Requirements volatility in multicultural situational contexts. In European Conference on Software Process Improvement (pp. 633–655). Springer International Publishing.
9. Siddique, I. (2024). Emerging trends in requirements engineering: A focus on automation and integration. *SSRN Electronic Journal.* doi: 10.2139/ssrn.4885927.
10. Tuunanen, T., Winter, R., & Brocke, J. V. (2024). Dealing with complexity in design science research: A methodology using design echelons. *MIS Quarterly, 48*(2), 427–458. doi: 10.25300/misq/2023/16700.
11. Yaşa, A., Özaltan, C. K., Ayten, G., Kaplama, F., Devran, Ö., Uçar, B. M., & Tüzün, E. (2025). Evaluating ReLink for traceability link recovery in practice. In Proc. 2025 IEEE Int. Conf. on Software Analysis, Evolution and Reengineering (SANER) (pp. 80–90).