From Scattered to Structured: A Vision for Automating Architectural Knowledge Management

Jan Keim
Karlsruhe Institute of Technology (KIT)
Karlsruhe, Germany
jan.keim@kit.edu

Angelika Kaplan
Karlsruhe Institute of Technology (KIT)
Karlsruhe, Germany
angelika.kaplan@kit.edu

Abstract

Software architecture is inherently knowledge-centric. The architectural knowledge is distributed across heterogeneous software artifacts such as requirements documents, design diagrams, code, and documentation, making it difficult for developers to access and utilize this knowledge effectively. Moreover, as systems evolve, inconsistencies frequently emerge between these artifacts, leading to architectural erosion and impeding maintenance activities. We envision an automated pipeline that systematically extracts architectural knowledge from diverse artifacts, links them, identifies and resolves inconsistencies, and consolidates this knowledge into a structured knowledge base. This knowledge base enables critical activities such as architecture conformance checking and change impact analysis, while supporting natural language question-answering to improve access to architectural knowledge. To realize this vision, we plan to develop specialized extractors for different artifact types, design a unified knowledge representation schema, implement consistency checking mechanisms, and integrate retrieval-augmented generation techniques for conversational knowledge access.

CCS Concepts

• **Software and its engineering** → *Development frameworks and environments*; **Software creation and management**; • **Computing methodologies** → **Natural language processing**; *Knowledge representation and reasoning*.

Keywords

Architectural Knowledge Management, Knowledge Extraction, Consistency Checking, Question-Answering Systems, Large Language Models

ACM Reference Format:
Jan Keim and Angelika Kaplan. 2026. From Scattered to Structured: A Vision for Automating Architectural Knowledge Management. In *2026 IEEE/ACM 48th International Conference on Software Engineering (ICSE-Companion '26), April 12–18, 2026, Rio de Janeiro, Brazil.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3774748.3795871

1 Introduction

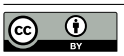
Software architecture constitutes a fundamental pillar of software engineering [32], embodying the high-level structure and design decisions that shape a system's quality attributes and long-term maintainability [22, 27]. However, architectural knowledge is inherently complex, drawing from both theoretical principles and practical experience accumulated throughout a system's lifecycle. [9, 17, 28] This knowledge manifests across different artifacts: requirements documents capture needs and constraints, design diagrams visualize system structures and component interactions, code comments embed implementation rationale, and documentation describes architectural decisions and their justifications.

The distribution of architectural knowledge across these heterogeneous artifacts presents significant challenges [9, 24]. While some knowledge is explicitly documented, substantial portions remain implicit, requiring inference from context or existing only in developers' minds [28]. Additionally, external sources contribute essential knowledge through established architectural styles, design patterns, and industry best practices. Accessing and effectively utilizing this scattered knowledge requires considerable effort from developers and architects, who must spend significant time and resources on comprehension [44], particularly in large-scale systems where documentation may span thousands of pages and codebases may contain millions of lines.

Compounding these accessibility challenges, inconsistencies frequently arise across artifacts as systems evolve [21, 43]. Documentation may fall out of sync with implementation, design diagrams may not reflect recent architectural changes, and code comments may describe outdated design decisions. Such inconsistencies breed confusion, impede maintenance activities, and can lead to architectural erosion, in which the implemented system gradually deviates from its intended design.

To address these challenges, we envision an approach (see Figure 1) that systematically extracts architectural knowledge from diverse software artifacts, identifies and resolves inconsistencies, and consolidates this knowledge into a structured knowledge base [23]. The resulting knowledge base can support critical activities, including architecture conformance checking, change impact analysis, and informed decision-making during architectural evolution. In addition to software artifacts like code and documentation, our long-term vision includes considering other sources, such as meeting minutes and recordings. Furthermore, by building upon structured knowledge representations, we can enable question-answering systems that allow developers and architects to query architectural knowledge through natural language, analogous to systems for scientific knowledge retrieval such as HubLink [19].

We also envision this system operating as an intelligent agent that continuously monitors software artifacts, automatically extracts knowledge, checks consistency across sources, and updates the knowledge base in real-time. When the automated analysis



This work is licensed under a Creative Commons Attribution 4.0 International License.
ICSE-Companion '26, Rio de Janeiro, Brazil
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2296-7/26/04
https://doi.org/10.1145/3774748.3795871

encounters ambiguities or conflicts that require domain expertise, the agent can proactively engage architects or developers through targeted questions, ensuring that human judgment informs critical decisions while minimizing the burden on development teams. Such capabilities promise to democratize access to architectural knowledge and support more informed software engineering practices throughout the development lifecycle. Additionally, this might encourage better documentation practices, as developers recognize the tangible benefits of maintaining accurate and consistent architectural knowledge.

The remainder of this paper is structured as follows: section 2 reviews the state-of-the-art and existing related work. section 3 outlines our planned steps to develop the automated knowledge extraction pipeline, including the agent-based monitoring approach, extraction framework, and knowledge base design. section 4 discusses anticipated challenges and open issues.

2 State-of-the-Art and Related Work

Managing and utilizing knowledge across various software artifacts is a fundamental challenge in software engineering. Throughout the development and maintenance lifecycle, substantial knowledge is embedded in diverse artifacts such as requirements documents, architectural models, source code, and test cases. Effectively leveraging this distributed knowledge and maintaining consistency across artifacts requires sophisticated techniques and tools. In the following subsections, we review key research areas that form the foundation for automated architectural knowledge management: Trace Link Recovery (TLR) for linking artifacts, consistency checking for detecting conflicts, knowledge representation approaches, the emerging role of large language models, and question-answering systems for knowledge access.

2.1 Traceability and Knowledge Integration

Trace links and their recovery (TLR) serve as a primary mechanism for identifying overlapping knowledge and integrating information scattered across different artifacts. These links support critical software engineering activities, including maintenance [5], bug localization [36], change impact analysis [8], and consistency checking [21, 42].

Researchers have developed diverse techniques for TLR, including information retrieval approaches [13, 15] and machine learning techniques [11, 34]. More recently, Large Language Models (LLMs) have emerged as a powerful tool for traceability [10, 16, 30, 38].

2.2 Consistency and Conformance Checking

Established trace links enable inconsistency detection across artifacts, as exemplified by Keim et al. [21]. Similarly, Bucaioni et al. [2] employ mappings to verify architectural conformance. Ensuring that implementations reflect their specifications and that knowledge remains consistent across all artifacts is crucial for software quality. Consequently, extensive research addresses conformance checking to verify that code and system behavior align with specifications, architectures, and other normative artifacts (e.g., [4, 29, 35, 39]).

The model-driven development community has particularly emphasized consistency across different models and system views, especially in complex domains such as cyber-physical systems [37]. Inconsistent models and views impede development, yet manually maintaining model consistency is challenging when numerous models coexist. Research in this area includes bidirectional model transformations [41], change-propagation mechanisms using consistency-preservation rules [26], and approaches that integrate multiple models into unified views [1, 3, 33].

2.3 Knowledge Bases and Knowledge Management

Knowledge bases provide structured approaches to support software engineering activities [7, 31]. Knowledge-based development facilitates systematic software evolution by explicitly capturing and utilizing domain and system knowledge. Ontologies offer a common representation mechanism for knowledge storage, leading to various ontology-based software engineering approaches [6]. Knowledge graphs extend these concepts by representing entities and their relationships as graphs, providing a powerful foundation for question-answering systems (cf. subsection 2.5) in which structured knowledge can be queried and combined with generative capabilities. Additionally, search-based software engineering techniques enable knowledge retrieval through queries [12].

2.4 Large Language Models in Software Engineering

LLMs have recently transformed software engineering practices through their capacity to process code and support diverse development tasks [14, 18]. Notably, LLMs can provide architectural knowledge and respond to developer queries [40], making them a promising technology for bridging knowledge across different artifacts and supporting automated knowledge management throughout the software lifecycle.

2.5 Question-Answering Systems

Question-answering (QA) systems enable users to retrieve specific information via natural language queries without requiring familiarity with the underlying data structures, making them valuable in software engineering contexts where developers need quick access to architectural knowledge and design decisions from large repositories. Knowledge graphs provide structured foundations for QA systems, while retrieval-augmented generation (RAG) approaches combine information retrieval with generative capabilities to produce accurate answers. Systems like HubLink, introduced by Kaplan et al. [19], demonstrate effective QA retrieval over knowledge graphs by combining graph-based retrieval with language-model capabilities. Adapting such approaches to software engineering knowledge bases could bridge the gap between scattered documentation and developers' information needs, supporting informed decision-making throughout the software lifecycle.

3 Planned Steps

To realize our vision, we will conduct a comprehensive literature review to further establish theoretical foundations, identify state-of-the-art approaches to architectural knowledge management and extraction from diverse artifacts, and derive a taxonomy of architectural knowledge types and an understanding of current limitations.

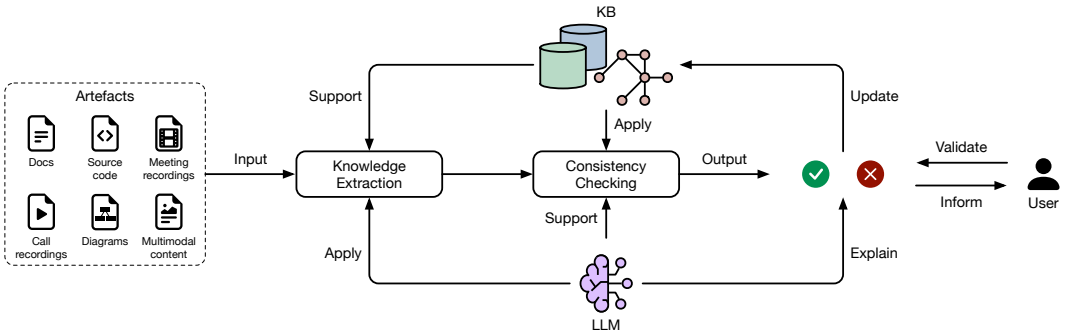


Figure 1: Artifacts knowledge extraction and consistency checking pipeline, by Keim et al. [23]

To realize the vision of an automated pipeline for architectural knowledge extraction and consolidation, we propose the following systematic approach, where we will empirically evaluate each step to ensure effectiveness and practicality:

3.1 Knowledge Extraction Framework

We will develop specialized extractors for different artifact types, each tailored to the unique characteristics of its source. Textual artifacts (e.g., requirements and documentation) will be processed using Natural Language Processing (NLP) techniques and LLMs to identify architectural elements, decisions, and rationale. Static analysis and program comprehension techniques will extract structural and behavioral knowledge from code artifacts. Visual artifacts such as design diagrams require parsing mechanisms to capture component relationships and architectural patterns. The expected outcome is a modular extraction framework capable of processing heterogeneous software artifacts and producing structured knowledge representations. To address inevitable gaps in automated extraction, such as missed key architectural terms, incorrectly inferred links, or project-specific domain knowledge, the framework will support Human-AI collaboration: Manual knowledge augmentation through user inputs (e.g., structured JSON documents) will allow users to provide missing information, correct extraction errors, and refine the knowledge base with project-specific context.

3.2 Knowledge Base Schema Design

We will design a unified schema for the architectural Knowledge Base (KB) that can accommodate knowledge from diverse sources while maintaining semantic relationships and traceability links through TLR. This schema has to support queries for architecture conformance checking, change impact analysis, and natural language question-answering. To link different artifacts, we will use traceability techniques from previous work [10, 20, 25].

3.3 Consistency Checking and Resolution

We will reuse existing and develop new automated mechanisms to identify inconsistencies between knowledge extracted from different artifacts. This includes detecting contradictions in architectural decisions, mismatches between documented and implemented structures, and outdated information. For example, in previous work, Keim et al. [21] demonstrated the use of traceability links for inconsistency detection, which we will adapt and extend for our multi-artifact context. We will explore rule-based, learning-based, and LLM-based approaches to inconsistency detection and develop strategies for semi-automated resolution that involve human architects in critical decisions.

3.4 Agent-Based Continuous Monitoring

To operationalize the knowledge management pipeline, we will develop an LLM agent that continuously monitors software artifacts and orchestrates the knowledge extraction and maintenance workflow. The agent will detect changes in artifacts through file system monitoring and version control integration, automatically triggering TLR for modified or newly created artifacts to maintain up-to-date traceability across the system. When inconsistencies are detected through automated consistency checking, the agent will analyze their severity and context, either resolving minor issues autonomously or escalating critical conflicts to human architects with targeted clarification requests. For ambiguous architectural decisions or conflicting information that requires domain expertise, the agent will formulate specific questions to architects or developers, presenting relevant context and suggesting potential resolutions. Before executing actions that significantly impact the knowledge base, such as removing outdated information or merging conflicting architectural decisions, the agent will seek explicit confirmation from responsible stakeholders. The expected outcome is an autonomous system that maintains knowledge base integrity with minimal human intervention while ensuring that critical decisions remain under human control.

3.5 Question-Answering System Integration

Building upon the structured KB, we will develop a natural language question-answering interface using Retrieval Augmented Generation (RAG) techniques. This will enable developers and architects to query architectural knowledge conversationally, similar to systems like HubLink [19] for scientific knowledge (see subsection 2.5). We will evaluate the system's ability to answer common architectural queries accurately and efficiently. The expected outcome is a prototype system that democratizes access to architectural knowledge across development teams.

4 Challenges

The envisioned approach presents several significant challenges that have to be addressed to ensure its effectiveness and reliability.

Heterogeneous artifact formats and multimodal data. Architectural knowledge manifests across diverse artifact types, structured sources such as code and formal documentation, partially structured sources such as design diagrams and models, and unstructured sources, including informal documentation, meeting transcripts, sketches, and informal notes. Processing the knowledge from these heterogeneous and multimodal sources requires techniques that can handle varying levels of formality and different representation formats without extensive manual preprocessing.

Scalability and context limitations. Large-scale systems may contain millions of lines of code and thousands of pages of documentation, which can challenge both computational resources and the context windows of LLMs used for knowledge extraction. The pipeline must efficiently process large artifact collections while maintaining accuracy, and determine optimal strategies for partitioning and processing artifacts that exceed model context limits. Therefore, techniques and approaches for effectively managing memory and context are essential.

Explainability and trust. LLM-based extraction approaches are susceptible to hallucinations, generating plausible but factually incorrect information. For architectural knowledge management, such errors can have serious consequences if they propagate into the knowledge base and influence critical decisions. Ensuring the explainability of extracted knowledge and building trust through verification mechanisms, like combining LLMs with formal reasoning tools and ontologies, is essential for practical adoption.

Distinguishing extraction errors from genuine inconsistencies. When the automated approach detects inconsistencies, it has to distinguish true conflicts in the artifacts (e.g., outdated documentation) from errors introduced during the extraction process. This distinction is crucial for determining appropriate resolution strategies and avoiding false alarms that burden developers. Furthermore, there may be planned inconsistencies, such as architectural changes that are not yet implemented, which the system must recognize and handle appropriately.

Human-AI collaboration and feedback integration. The agent-based approach has to strike an appropriate balance between automation and human oversight. Excessive automation without sufficient human validation may introduce errors, while excessive human intervention negates efficiency benefits. Designing effective interaction patterns for clarification requests, confirmation workflows, and feedback integration requires careful consideration of developer workflows and cognitive load.

Contextual learning and project-specific adaptation. Each software project exhibits unique architectural patterns, naming conventions, and documentation practices. The knowledge extraction pipeline has to adapt to project-specific contexts using techniques such as few-shot learning and incremental refinement guided by developer feedback, rather than relying solely on general-purpose models trained on external data.

Knowledge base performance and maintenance. As the knowledge base grows with continuous artifact monitoring and extraction, maintaining query efficiency becomes critical to support real-time architecture conformance checking and responsive question-answering systems. The pipeline has to implement effective update management strategies to handle incremental knowledge additions while ensuring consistency, managing versioning, and avoiding performance degradation. Furthermore, when processing sensitive artifacts such as meeting minutes or voice recordings, data privacy considerations require appropriate access controls and anonymization techniques.

Temporal validity and system evolution. Software systems evolve continuously, and some evolutionary steps are dramatic, such as migrating to entirely different architecture styles and technology stacks. Despite these transformations, the system remains conceptually the same entity, yet its architectural form changes fundamentally. This creates a significant challenge for knowledge management: architectural knowledge that is valid at one time may become obsolete or incorrect at another time, yet it represents important historical context. The approach must address whether to discard outdated information, thereby losing valuable historical insights and evolution rationale, or to maintain a temporal knowledge base with versioning and time-aware query capabilities. This requires sophisticated mechanisms to track changes over time and ensure that queries return contextually relevant information based on the system's state at specific points in its evolution.

Security and Access Management. When processing sensitive artifacts such as meeting minutes or voice recordings, data privacy considerations require appropriate access controls and anonymization techniques. Moreover, ensuring that the knowledge extraction and storage processes comply with organizational security policies is essential to protect sensitive architectural information from unauthorized access or breaches.

Acknowledgments

This work was funded by the Topic Engineering Secure Systems of the Helmholtz Association (HGF) and by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under the National Research Data Infrastructure – NFDI 52/1 – project number 501930651, NFDIxCS. It was also supported by funding from the pilot program Core Informatics at KIT (KiKIT) of the Helmholtz Association (HGF) and by KASTEL Security Research Labs.

References

[1] Colin Atkinson and Dietmar Stoll. 2008. Orthographic modeling environment. In *Proceedings of the Theory and Practice of Software, 11th International Conference on Fundamental Approaches to Software Engineering* (Budapest, Hungary) *(FASE'08/ETAPS'08)*. Springer-Verlag, Berlin, Heidelberg, 93–96.

[2] Alessio Bucaioni, Amleto Di Salle, Ludovico Iovino, Leonardo Mariani, and Patrizio Pelliccione. 2024. Continuous Conformance of Software Architectures. In *2024 IEEE 21st International Conference on Software Architecture (ICSA)*. 112–122. doi:10.1109/ICSA59870.2024.00019

[3] Erik Johannes Burger. 2013. Flexible views for view-based model-driven development. In *Proceedings of the 18th International Doctoral Symposium on Components and Architecture* (Vancouver, British Columbia, Canada) *(WCOP '13)*. ACM, New York, NY, USA, 25–30. doi:10.1145/2465498.2465501

[4] Selim Çiraci, Hasan Sözer, and Bedir Tekinerdogan. 2012. An Approach for Detecting Inconsistencies between Behavioral Models of the Software Architecture and the Code. In *IEEE 36th Annual Computer Software and Applications Conference*. 257–266. doi:10.1109/COMPSAC.2012.36

[5] Jane Cleland-Huang. 2012. *Traceability in Agile Projects*. Springer London, London, 265–275. doi:10.1007/978-1-4471-2239-5_12

[6] T. S. Dillon, E. Chang, and P. Wongthongtham. 2008. Ontology-Based Software Engineering- Software Engineering 2.0. In *19th Australian Conference on Software*

Engineering (aswec 2008). 13–23. doi:10.1109/ASWEC.2008.4483185

[7] Wei Ding, Peng Liang, Antony Tang, and Hans van Vliet. 2014. Knowledge-based approaches in software documentation: A systematic literature review. *Information and Software Technology* 56, 6 (2014), 545–567. doi:10.1016/j.infsof.2014.01.008

[8] Davide Falessi, Justin Roll, Jin L.C. Guo, and Jane Cleland-Huang. 2020. Leveraging Historical Associations between Requirements and Source Code to Identify Impacted Classes. *IEEE Transactions on Software Engineering* 46, 4 (2020), 420–441. doi:10.1109/TSE.2018.2861735

[9] R. Farenhorst and H. van Vliet. 2009. Understanding how to support architects in sharing knowledge. In *ICSE Workshop on Sharing and Reusing Architectural Knowledge*. 17–24. doi:10.1109/SHARK.2009.5069111

[10] Dominik Fuchß, Tobias Hey, Jan Keim, Haoyu Liu, Niklas Ewald, Tobias Thirolf, and Anne Koziolek. 2025. LiSSA: Toward Generic Traceability Link Recovery through Retrieval-Augmented Generation. In *Proceedings of the IEEE/ACM 47th International Conference on Software Engineering* (Ottawa, Canada) *(ICSE '25)*. Institute of Electrical and Electronics Engineers (IEEE).

[11] Jin Guo, Jinghui Cheng, and Jane Cleland-Huang. 2017. Semantically Enhanced Software Traceability Using Deep Learning Techniques. In *Proceedings of the 39th International Conference on Software Engineering (ICSE '17)*. IEEE Press, Piscataway, NJ, USA, 3–14. doi:10.1109/ICSE.2017.9

[12] Mark Harman, S. Afshin Mansouri, and Yuanyuan Zhang. 2012. Search-based software engineering: Trends, techniques and applications. *ACM Comput. Surv.* 45, 1, Article 11 (Dec. 2012), 61 pages. doi:10.1145/2379776.2379787

[13] Jane Huffman Hayes, Alex Dekhtyar, and Senthil Karthikeyan Sundaram. 2006. Advancing candidate link generation for requirements tracing: the study of methods. *IEEE Transactions on Software Engineering* 32, 1 (2006), 4–19. doi:10.1109/TSE.2006.3

[14] Junda He, Christoph Treude, and David Lo. 2025. LLM-Based Multi-Agent Systems for Software Engineering: Literature Review, Vision and the Road Ahead. *ACM Trans. Softw. Eng. Methodol.* (Jan. 2025). doi:10.1145/3712003 Just Accepted.

[15] Tobias Hey, Fei Chen, Sebastian Weigelt, and Walter F. Tichy. 2021. Improving Traceability Link Recovery Using Fine-grained Requirements-to-Code Relations. In *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)* (2021-09). 12–22. doi:10.1109/ICSME52107.2021.00008

[16] Tobias Hey, Dominik Fuchß, Jan Keim, and Anne Koziolek. 2025. Requirements Traceability Link Recovery via Retrieval-Augmented Generation. In *Proceedings of the 31st International Working Conference on Requirement Engineering: Foundation for Software Quality*. doi:10.5445/IR/1000178589

[17] A. Jansen and J. Bosch. 2005. Software Architecture as a Set of Architectural Design Decisions. In *5th Working IEEE/IFIP Conference on Software Architecture (WICSA'05)*. 109–120. doi:10.1109/WICSA.2005.61

[18] Haolin Jin, Linghan Huang, Haipeng Cai, Jun Yan, Bo Li, and Huaming Chen. 2024. From LLMs to LLM-based Agents for Software Engineering: A Survey of Current, Challenges and Future. arXiv:2408.02479 [cs.SE] https://arxiv.org/abs/2408.02479

[19] Angelika Kaplan, Jan Keim, Marco Schneider, and Ralf Reussner. 2025. HubLink: A Novel Question Answering Retrieval Approach over Knowledge Graphs. In *RAGE-KG 2025: The Second International Workshop on Retrieval-Augmented Generation Enabled by Knowledge Graphs (RAGE-KG 2025) co-located with 24th International Semantic Web Conference (ISWC 2025). Ed.: D. Dobrij (CEUR Workshop Proceedings, Vol. 4079)*. CEUR-WS, 1–18.

[20] Jan Keim, Sophie Corallo, Dominik Fuchß, Tobias Hey, Tobias Telge, and Anne Koziolek. 2024. Recovering Trace Links Between Software Documentation And Code. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering* (Lisbon, Portugal) *(ICSE '24)*. ACM, New York, NY, USA, Article 215, 13 pages. doi:10.1145/3597503.3639130

[21] Jan Keim, Sophie Corallo, Dominik Fuchß, and Anne Koziolek. 2023. Detecting Inconsistencies in Software Architecture Documentation Using Traceability Link Recovery. In *2023 IEEE 20th International Conference on Software Architecture (ICSA)*. 141–152. doi:10.1109/ICSA56044.2023.00021

[22] Jan Keim, Tobias Hey, Bjarne Sauer, and Anne Koziolek. 2023. A Taxonomy for Design Decisions in Software Architecture Documentation. In *Software Architecture. ECSA 2022 Tracks and Workshops*. Springer International Publishing, Cham, 439–454.

[23] Jan Keim, Tobias Hey, Vincenzo Scotti, Raffaela Mirandola, and Anne Koziolek. 2025. *Towards Automated Knowledge Management in the Software Life Cycle*. Technical Report. Karlsruher Institut für Technologie (KIT). doi:10.5445/IR/1000181618

[24] Jan Keim and Anne Koziolek. 2019. Towards Consistency Checking Between Software Architecture and Informal Documentation. In *2019 IEEE ICSA Companion*. 250–253. doi:10.1109/ICSA-C.2019.00052

[25] Jan Keim, Sophie Schulz, Dominik Fuchß, Claudius Kocher, Janek Speit, and Anne Koziolek. 2021. Trace Link Recovery for Software Architecture Documentation. In *Software Architecture*. Springer International Publishing, Cham, 101–116.

[26] Max E. Kramer, Michael Langhammer, Dominik Messinger, Stephan Seifermann, and Erik Burger. 2015. Change-Driven Consistency for Component Code, Architectural Models, and Contracts. In *Proceedings of the 18th International ACM SIGSOFT Symposium on Component-Based Software Engineering* (Montréal, QC, Canada) *(CBSE '15)*. ACM, New York, NY, USA, 21–26. doi:10.1145/2737166.2737177

[27] Philippe Kruchten. 2004. An ontology of architectural design decisions in software intensive systems. In *2nd Groningen workshop on software variability*. Citeseer, 54–61.

[28] Philippe Kruchten, Patricia Lago, and Hans van Vliet. 2006. Building Up and Reasoning About Architectural Knowledge. In *Quality of Software Architectures (Lecture Notes in Computer Science)*. Springer, 43–58. doi:10.1007/11921998_8

[29] J Jenny Li and Joseph R Horgan. 1998. To maintain a reliable software specification. In *International Symposium on Software Reliability Engineering*. IEEE, 59–68.

[30] Jinfeng Lin, Yalin Liu, Qingkai Zeng, Meng Jiang, and Jane Cleland-Huang. 2021. Traceability Transformed: Generating More Accurate Links with Pre-Trained BERT Models. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. 324–335. doi:10.1109/ICSE43902.2021.00040

[31] Michael R. Lowry. 1993. Methodologies for knowledge-based software engineering. In *Methodologies for Intelligent Systems*, Jan Komorowski and Zbigniew W. Raś (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 219–234.

[32] Nenad Medvidovic and Richard N. Taylor. 2010. Software architecture: foundations, theory, and practice. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 2* (Cape Town, South Africa) *(ICSE '10)*. ACM, New York, NY, USA, 471–472. doi:10.1145/1810295.1810435

[33] Johannes Meier, Heiko Klare, Christian Tunjic, Colin Atkinson, Erik Burger, Ralf Reussner, and Andreas Winter. 2019. Single Underlying Models for Projectional, Multi-View Environments. In *Proceedings of the 7th International Conference on Model-Driven Engineering and Software Development - MODELSWARD*. INSTICC, SciTePress, 119–130. doi:10.5220/0007396401190130

[34] C. Mills, J. Escobar-Avila, and S. Haiduc. 2018. Automatic Traceability Maintenance via Machine Learning Classification. In *2018 IEEE (ICSME)*. 369–380. doi:10.1109/ICSME.2018.00045

[35] Leo J. Pruijt, Christian Köppe, Jan Martijn van der Werf, and Sjaak Brinkkemper. 2014. HUSACCT: architecture compliance checking with rich sets of module and rule types. In *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering* (Vasteras, Sweden) *(ASE '14)*. ACM, 851–854. doi:10.1145/2642937.2648624

[36] Michael Rath, David Lo, and Patrick Mäder. 2018. Analyzing Requirements and Traceability Information to Improve Bug Localization. In *Proceedings of the 15th International Conference on Mining Software Repositories* (Gothenburg, Sweden) *(MSR '18)*. ACM, New York, NY, USA, 442–453. doi:10.1145/3196398.3196415

[37] Ralf Reussner, Ina Schaefer, Bernhard Beckert, Anne Koziolek, and Erik Burger. 2023. Consistency in the View-Based Development of Cyber-Physical Systems (Convide). In *2023 ACM/IEEE MODELS Companion*. 83–84. doi:10.1109/MODELS-C59198.2023.00026

[38] Alberto D. Rodriguez, Katherine R. Dearstyne, and Jane Cleland-Huang. 2023. Prompts Matter: Insights and Strategies for Prompt Engineering in Automated Software Traceability. In *2023 IEEE 31st International Requirements Engineering Conference Workshops (REW)*. 455–464. doi:10.1109/REW57809.2023.00087

[39] Sandra Schröder and Matthias Riebisch. 2018. An Ontology-Based Approach for Documenting and Validating Architecture Rules. In *Proceedings of the 12th European Conference on Software Architecture: Companion Proceedings (ECSA '18)*. ACM, New York, NY, USA, Article 52. doi:10.1145/3241403.3241457

[40] Mohamed Soliman and Jan Keim. 2025. Do Large Language Models Contain Software Architectural Knowledge? An Exploratory Case Study with GPT. In *22nd IEEE International Conference on Software Architecture (ICSA 2025)*. Institute of Electrical and Electronics Engineers (IEEE).

[41] Perdita Stevens. 2008. *A Landscape of Bidirectional Model Transformations*. Springer Berlin Heidelberg, Berlin, Heidelberg, 408–424. doi:10.1007/978-3-540-88643-3_10

[42] Nataliia Stulova, Arianna Blasi, Alessandra Gorla, and Oscar Nierstrasz. 2020. Towards Detecting Inconsistent Comments in Java Source Code Automatically. In *2020 IEEE 20th International Working Conference on Source Code Analysis and Manipulation (SCAM)*. 65–69. doi:10.1109/SCAM51674.2020.00012

[43] Rebekka Wohlrab, Ulf Eliasson, Patrizio Pelliccione, and Rogardt Heldal. 2019. Improving the Consistency and Usefulness of Architecture Descriptions: Guidelines for Architects. In *2019 IEEE ICSA*. 151–160. doi:10.1109/ICSA.2019.00024

[44] Xin Xia, Lingfeng Bao, David Lo, Zhenchang Xing, Ahmed E. Hassan, and Shanping Li. 2018. Measuring Program Comprehension: A Large-Scale Field Study with Professionals. *IEEE Transactions on Software Engineering* 44, 10 (2018), 951–976. doi:10.1109/TSE.2017.2734091