Towards Automated Knowledge Management in the Software Life Cycle

JAN KEIM, Karlsruhe Institute of Technology, Germany
TOBIAS HEY, Karlsruhe Institute of Technology, Germany
VINCENZO SCOTTI, Karlsruhe Institute of Technology, Germany
RAFFAELA MIRANDOLA, Karlsruhe Institute of Technology, Germany
ANNE KOZIOLEK, Karlsruhe Institute of Technology, Germany

Artificial Intelligence (AI), in particular that built upon Large Language Models (LLMs), is reshaping the way we approach many processes. Mainly thanks to the introduction of coding assistants, software development has been entangled in this transformation as well. However, when we talk about software development, coding is just a part of this intricate process, and it is not the only one that can benefit from the support of AI. In fact, LLMs are already being successfully applied to other stages of the software life cycle, such as requirement analysis and design. With this paper, we focus on the big picture of the entire life cycle of software and on maintaining structured knowledge about the system, with attention on maintaining the consistency with respect to formal and informal specifications across all artefacts. The core of our idea is to involve LLM-based assistants within all the life cycle stages, where information to keep track of can come from both well-structured artefact sources (e.g., in source code and documentation) as well as sparse and unstructured ones (e.g., in meeting transcripts). In fact, these assistants can potentially be used to automate information extraction from multiple different sources and to maintain a Knowledge Base (KB) modelling the knowledge about the system that can be used to ensure continuous consistency of the project artefacts through validation against the KB itself. The role we envision for AI is thus (i) to bridge the gap between structured and unstructured knowledge and its formal representation and (ii) to automate the update of information and the verification of consistency.

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

Additional Key Words and Phrases: AI4SE, LLM, NLP, Consistency, Software life cycle, Knowledge base

ACM Reference Format:
Jan Keim, Tobias Hey, Vincenzo Scotti, Raffaela Mirandola, and Anne Koziolek. 2025. Towards Automated Knowledge Management in the Software Life Cycle. In *Future Software Engineering Workshop 2025 (FSEws2025)*. 11 pages. https://doi.org/10.5445/IR/1000181618

1 Introduction

Software development is undergoing an impressive transformation driven by the adoption of Large Language Models (LLMs) [51] as support for the development process. These Artificial Intelligence (AI) tools resulting from Natural Language Processing (NLP) and *Deep Learning* research have quickly spread across many domains thanks to their flexibility and robustness. Probably, the major breakthrough of LLMs in software development has been that of *coding assistants*, like *Copilot* [22]. However, LLM-based approaches have already been used successfully in the other stages of

Authors' Contact Information: Jan Keim, jan.keim@kit.edu, Karlsruhe Institute of Technology, Karlsruhe, Germany; Tobias Hey, hey@kit.edu, Karlsruhe Institute of Technology, Karlsruhe, Germany; Vincenzo Scotti, vincenzo.scotti@kit.edu, Karlsruhe Institute of Technology, Karlsruhe, Germany; Raffaela Mirandola, raffaela.mirandola@kit.edu, Karlsruhe Institute of Technology, Karlsruhe, Germany; Anne Koziolek, koziolek@kit.edu, Karlsruhe Institute of Technology, Karlsruhe, Germany.

the software life cycle, such as *requirement engineering* [2, 29], *software design* [16, 71], *software quality assurance* [66, 75], *software maintenance* [19, 36], and *software management* [30].

Those approaches mainly address a specific software engineering tasks to be solved. However in 2030 we believe LLMs will be able to support the whole software development life cycle. The software life cycle as a whole can benefit by maintaining the consistency across all artefacts throughout the entire lifecycle [39]. Maintaining this consistency can become challenging as taking care of always updating the knowledge about the system and validating every new artefact against this knowledge gets tedious and time-consuming. However, properly managing all the knowledge is fundamental to ensure a consistent evolution of the software itself. As a result, we often reach a point where inconsistencies arise between the intended model and the generated artefacts resulting in a degradation of the software. Nevertheless, most of the information is available, whether we consider that coming from informal and unstructured data (e.g., meeting transcripts) or more structured one (e.g., source code).

Keeping knowledge documented in structured data consistent (e.g., source code and documentation [73]) is by itself a challenging task. But also informal sources (e.g., meeting minutes or sketches) are relevant for defining the system and vice versa expressions of these sources can be used to identify outdated or inconsistent knowledge in the mental models of the individuals involved.

However, we believe that the next generations of LLMs will ease the consistent processing of those different types of artifacts and data, and make an automated knowledge management across the entire life cycle possible. We propose to exploit LLM in conjunction with data coming from all artefacts to

(1) automatically manage knowledge and
(2) check consistency of artefacts against such knowledge

throughout the entire life cycle by automatically maintaining and interfacing with a Knowledge Base (KB), as we depict in Figure 1. The idea is to have the LLM continuously interacting with the KB at each stage to integrate new knowledge coming from the different artefacts and, at the same time, use the KB to provide feedback about consistency of artefacts with the intended model encoded in the KB.

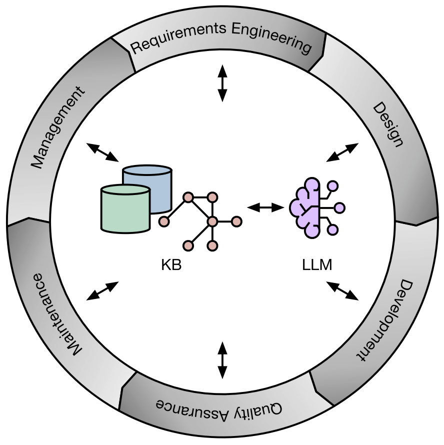

Fig. 1. LLM and KB for continuous consistency.

Thanks to the advances introduced by LLMs, we can now have a single tool managing the information extraction from unstructured and structured data in an end-to-end fashion. Moreover, the same tool can be used to interface the KB both for inserting and updating information, for running queries about consistency, and to process and report responses of such queries as well. Additionally, thanks to the flexibility given by multi-modal models, we can now integrate information extracted from different sources, whether they are the raw audio recordings of a meeting or images with the sketches about the architecture appearing in some notes.

As a motivating application scenario we will focus in this paper on software architecture knowledge, as it has effects on most other stages of the life cycle. We divide this paper into the following sections. In Section 2, we report on existing approaches in modelling and validating knowledge about software architecture. In Section 3, we outline our long-term vision of how LLMs can be used to automate knowledge management in software life cycle stages. In Sections 4 and 5, we comment, respectively, on the research issues and research directions connected to the expected challenges for Software Engineering (SE) in the near future [55]. In Section 6, we summarise our views and proposals.

2 Related Work

There are various ways to use the knowledge that is contained within the various artefacts that are produced during the development and maintenance of a software system, and to check for and deal with potential inconsistencies.

Trace links are commonly used to identify overlapping knowledge and combine the knowledge that is scattered in the different artefacts. These links help in various activities like software maintenance [10, 46, 47], bug localisation [59], change impact analyses [15, 47], and consistency analyses [38, 39, 70].

There are various techniques to create these trace links. Some approaches are based on information retrieval techniques such as vector space models [25], word embeddings [27], probabilistic models [54], or clustering [56]. Other approaches use machine learning and use techniques like trained neuronal networks [23], classifiers [50], or genetic algorithms [63]. Recent research also applies LLMs, like the work by Lin et al. [43], Rodriguez et al. [62], Hey et al. [28], or Fuchß et al. [19].

The found trace links can be used to detect inconsistencies, as the work by Keim et al. [38] demonstrates. Similarly, Bucaioni et al. [6] use mappings to check the conformance to a given architecture. Conformance checking is important to ensure that the knowledge about a software system is consistent across all artifacts and that the implementation reflects the knowledge and specification. As such, there are various further works that deal with conformance checking to ensure that code or a system's behavior conforms to a specification, architecture or similar (e.g., [8, 42, 57, 65]).

The model-driven development community also identified consistency between different models and views of the system as an important challenge, especially for example for cyber-physical systems [61]. Deviating models and views negatively influence the development, but manually keeping models consistent is a challenging task, in particular in some domains where plenty of different models are used. Therefore, there is various related work that concerns bidirectional transformations between models [69], propagating changes from one model to related models to ensure consistency using consistency preservation rules [41], and approaches that try to combine multiple models in a way that users can view them as a single underlying model [4, 7, 48].

There are also different works and activities that are related to the use of knowledge bases to support software engineering activities [13, 44]. A knowledge-based approach facilitates the development and combines it with the use of knowledge to ensure a successful software development and evolution. Ontologies are a common way to store and represent knowledge and, thus, there are different ontology-based approaches in software engineering [12]. Furthermore, knowledge can also be retrieved from search queries, and there are various works that deal with search-based software engineering [24].

Lastly, LLMs have found their way into software engineering due to their ability to handle code and overall support software engineering tasks [26, 35]. For example, LLMs have shown the ability to provide information about architectural knowledge and answer corresponding questions of developers [68]. As such, LLMs are a promising approach to support software engineering activities and to provide knowledge across different artefacts.

3 AI-assisted Software Development

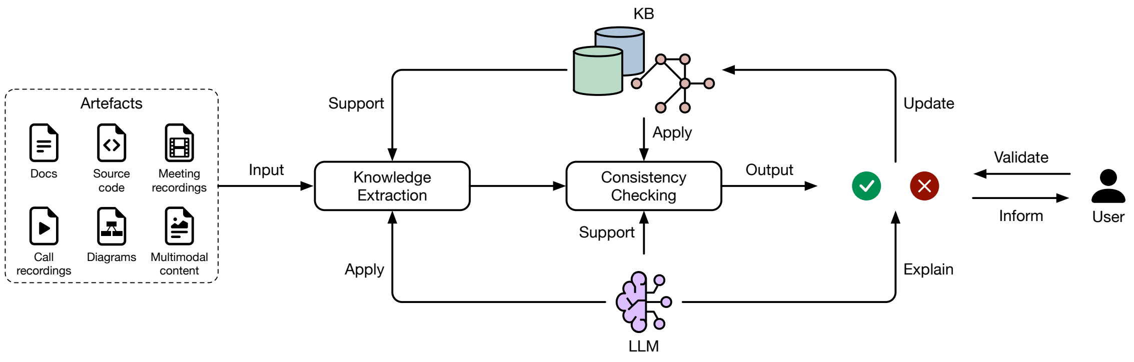

Fig. 2. Artefacts knowledge extraction and consistency checking pipeline.

In this section we present our view for the future of AI-assisted software development. We provide an overview of the use of AI during the software lifecycle as companion at each stage (Section 3.1), then we provide better details on the aspects related to knowledge extraction from the different sources (Section 3.2), consistency checking to ensure that the evolution is consistent with the intended model architecture (Section 3.3).

3.1 Overview

The software life cycle involves the generation of many different artefacts. These artefacts cover different domains and modalities: they go from informal and unstructured documents describing the requirements, to more structured documents as the architecture design and, eventually, to well structured source code. As the project evolves, maintaining the consistency between the artefacts and the intended architecture is critical, especially when the software starts evolving and changing. To this end, it is fundamental to keep track of all the information concerning the architecture to be sure every new artefact is compliant with the rest of the project.

Our proposed methodology tackles exactly the problem of automatically maintaining knowledge about the system as its life cycle proceeds and, at the same time, ensure consistency with respect to such knowledge. For the scope, we propose to use a combination of LLM-based AI and KB, as depicted in Figure 2. LLMs offer the degree of flexibility necessary to have a single tool interface with users, KBs, and to process artefacts spanning different domains, while the KB – based on an ontology or other formal specifications – offers the soundness of logical reasoning, which LLMs lack, at the cost of a less friendly interface, which is managed by the LLM. The idea is to reduce the burden of manual annotation labour of developers, as well as any other involved user, in the development life cycle.

3.2 Knowledge extraction

The issue of extracting structured information from unstructured text data is an ongoing problem in NLP, known with the name of *Information Extraction* [37]. Traditionally, the problem was decomposed into smaller ones (e.g., *Named Entity Recognition* and *Relation Extraction*) to extract step-by-step pieces of knowledge to feed a KB.

The advent of *Transformer*-based language models [11, 58, 72] that could be fine-tuned to solve specific language-related tasks has moved forward results on this topic as well [14]. Nowadays, with LLMs, we can approach the problem in an end-to-end fashion, generating data directly in structured format (e.g., CSV or JSON), or we can specify a step-by-process to follow for more difficult problems [17]. Moreover, as we detail in Section 5.3, LLMs are capable of handling different information sources, whether they are text documents or source code. In the last couple of years we have also witnessed the evolution towards multi-modal data processing of LLMs, allowing to process images, audio, and video as well.

The crucial point concerning knowledge extraction is to integrate knowledge about the KB into the LLM to make sure it will extract all relevant information from the artefacts without extracting irrelevant information. Given the potential size of processed artefacts (e.g., entire documents for requirements analysis, or portions of the code base) compared to the limited context window of some LLMs, finding the optimal combination yielding best performances is non-trivial. Given the absence of large data sets for fine-tuning, we envision to use in-context learning, adapting the LLM step-by-step via prompting. Moreover, a further research challenge would be to have the LLM do the work from scratch, defining the metamodels or the ontologies necessary to set up the KB instead of leveraging user provided structures.

3.3 Consistency Checking

Consistency check is the other side of the coin. While the LLM finds its use in bridging the artefacts with the KB pushing the extracted information, the KB takes care of maintaining the consistency by ensuring that none of the new information violates constraints or is in contrast with already stored knowledge through sound verification.

There are various challenging parts when checking consistency. Inconsistencies can arise from actual errors in the artefacts, but also from faults in the information extraction process, either from the current artefact or from some previously processed one, or there can be faults in the specification of the KB. As we detail in Sections 4.1 and 5.1, explainability will play a crucial role in tracing these faults.

4 Issues

In this section, we discuss the research issues related to integrating AI assistants based on LLMs into the software life cycle to manage the knowledge about the architecture and to validate the consistency of artefacts with respect to this knowledge. The discussion spans explainability (Section 4.1), defects – in the sense of inconsistencies – (Section 4.2), knowledge integration (Section 4.3) and human-AI collaboration (Section 4.4).

4.1 Explainability and Trust

Current AI-based applications lack reliability (intended as the "ability of an item to perform as required, without failure" [1]). In the case of LLMs, this problem is mostly due to *hallucinations* [31, 34], a problem related to the generation content that is neither *factual* with respect to common world knowledge nor *faithful* with respect to the current input, which harms LLMs *trustworthiness* [32].

LLMs are not designed for sound logical reasoning, thus they cannot be used to directly assess consistency reliably. Assessing consistency is something that can be delegated to formal tools like an ontology underlying the KB, while the

LLM takes care of interfacing with the KB. We can better exploit the capability of LLMs for extrapolating patterns from text and the vast world knowledge embedded in their weights in conjunction with techniques like few-shot learning [5, 9, 64] and Retrieval Augmented Generation (RAG) [21] to increase the likelihood of extracting the correct information from an analysed artefact to be validated against the KB. Moreover, the adoption of *explainability* techniques to trace the sources of LLM predictions would help transparency and trustworthiness about model predictions, giving developers a reference to confirm or disprove such prediction (generating also a valuable feedback for the LLM).

4.2 Defect Prediction

Defect prediction is yet another issue that can be tackled with the help of AI. In this paper we focus on the problem of detecting defects uprising from inconsistencies between the intended model and the generated artefacts, which we suggest to tackle in a passive way. Every time a new artefact is generated, an AI agent can process it automatically to extract relevant information and match it against the knowledge base. The challenge is to integrate data from the artifact and the structure of the KB in the LLM input to effectively extract relevant information. This also gives us an interesting research direction connected to contextual learning (see Section 5.2).

From an application perspective, a simple solution would be to trigger a warning whenever an inconsistency is found. However, we also envision more sophisticated approaches where an LLM can be used to suggest a possible fix for the inconsistency. For example, generating coherent documentation or updating an existing one while sticking to the context and tone of the existing documentation was infeasible, but this is one of the cases where LLMs can really shine. An advanced approach would be to get this process to work in real-time, as the artefact is generated, to immediately spot the inconsistency. For example, inform the participants of meeting that their discussion does not adhere to the taken design decisions of the system.

4.3 Feedback Loops and Knowledge Integration

Feedback management and knowledge integration are deeply entangled in the methodology we propose. In fact, the core role of AI in the software life cycle will be to maintain knowledge, taking care of processing all artefacts to extract relevant information that can be used to extend the KB about the software being developed and, in case of conflicts arising from the KB itself, detect the conflict and notify the users. At the same time, information about the KB and its content needs to be integrated in the prompt of the LLM to ensure that the relevant information is extracted from raw data.

Artefacts processing needs to be validated by the developers, to ensure that all and only the correct information is added to the KB and that detected inconsistencies are concrete problems. All these feedbacks can be used to improve the system, generating useful examples to either refine the models or use as references for following predictions of the LLM. This leads to more intricate challenges involving autonomous error correction by the LLM based on the KB or the developer response about consistency [67], like automatically updating a piece of code to fix consistency issues with respect to some requirements, or, at least, suggest a possible fix the developers can validate.

4.4 Human-AI Collaboration

A crucial point will be to make users (i.e., developers) understand how valuable this AI support is in reducing the manual labour expected from them. In fact, the adoption of AI in software development is mainly thought to support developers and speed-up the process, rater than replacing them. At the same time, the feedbacks provided by the users on LLM predictions will be fundamental to improve the AI.

The performance of the LLM will improve with the amount of feedback from the users validating LLMs the explanations, the knowledge extraction, and the results of the consistency checking. Similarly, the resulting improved predictions by the LLM can reduce the human effort in the pipeline. Overall, there is a mutual cooperation and a spiral of ongoing improvements.

5 Directions

In this section, we present the research directions we are willing to explore with the proposed approach to adopt AI for consistency checking, which cover the integration of AI in SE (Section 5.1), the contextual learning (Section 5.2) and the exploration of new domains for AI in SE (Section 5.3).

5.1 Bridging AI and SE

Integrating AI in the software life cycle introduces new paradigms of development, where the human developers are empowered by an automatic tool capable of tracking the consistency with the intended product at each stage. Nevertheless, humans will stay at the centre of each process: AI will reduce the burden on manual labour to update and validate the KB and at the same time humans will supervise AI. *Explainability* will allow transparency and trustworthiness in these human-AI interactions.

In fact, explainability will play a fundamental role at each stage, connecting LLM predictions about consistency with references in the artefacts and constraints violated in the knowledge base. At the same time, explanations are not only a mean to justify an inconsistency, but also a way to improve trustworthiness in the generated knowledge to be added in the KB by tracking the sources of the new information. However, it will always be up to humans to responsibly validate these predictions, producing valuable feedback.

5.2 Contextual Learning

Each software project has unique characteristics that require AI to adapt as the development process continues. The major takeaway from the first LLM results is how well they can perform in-context (i.e., few-shot) learning (i) to improve task understanding and generalisation [5, 9, 64] and (ii) to integrate additional information besides the static one memorised in their weights [21, 40]. Moreover, through the techniques to extend model contexts [60, 76], there are LLMs capable of processing large documents with sparse information, as shown by the results on the *Needle in a Haystack* benchmark [52].

Thus, we are interested in exploiting all these capabilities to make the AI supporting the development more aware of the current state of the project. To make the AI actually extrapolate the knowledge from the different sources, we need to provide the information about the knowledge base and the source being processed, making the model ingest a lot of different information. What we want to understand is how to do it practically and to which extent an approach can deal with all this information in an end-to-end fashion, without additional support.

5.3 New Domains

The diversity of documents scraped from the web and used to pre-train the LLMs underlying AI-based applications makes them compatible with a huge variety of domains, whether it is document format (e.g., plain text or source code) or topics (e.g., news or technical documents) [20]. Nowadays, thanks to the advent of multi-modal models, we can integrate different sources of information. There are a variety of such models, including either closed-access ones, like *GPT* or *ChatGPT* [33, 53] and *Gemini* [3, 60], or open-access ones, like *DeepSeek* [45, 74] and *Llama* [49]. In fact, state-of-the-art models can ingest text as well as images, video and audio, extending significantly the range of domains to extract relevant information from.

The analysis will not be limited to sequences of text coming from documents redacted by human or from source code. Current models can process video recordings of meetings or handwritten notes with sketches of architecture diagrams [18]. Moreover, experiments on the *Needle in a Haystack* benchmark are showing how good LLMs are in extracting information not only from incredibly long contexts, but from very diverse ones involving multiple modalities, making this a good chance to gather results from real applications of long multimodal contexts processing. We found these opportunities valuable to understand how much we can exploit AI to automate information extraction from raw multi domain data (intended as multiple source modalities, format and topics) without any preprocessing.

6 Conclusion

With this vision paper, we proposed our view on advancing the approach to software development by integrating AI in the different steps of the life cycle. The key idea is to exploit LLMs flexibility to extract and combine knowledge from different sources and modalities to process all artefacts generated during the life cycle with the soundness of a KB to keep track of this knowledge and validate the consistency at each step. We expect that using AI-based tools to take care of this laborious step will improve software quality and development speed.

Acknowledgments

This work was supported by funding from the pilot program Core Informatics at KIT (KiKIT) of the Helmholtz Association (HGF) and supported by the German Research Foundation (DFG) - SFB 1608 - 501798263 and KASTEL Security Research Labs, Karlsruhe.

References

[1] NIST AI. 2023. Artificial Intelligence Risk Management Framework (AI RMF 1.0). https://doi.org/10.6028/NIST.AI.100-1

[2] Waad Alhoshan, Alessio Ferrari, and Liping Zhao. 2023. Zero-shot learning for requirements classification: An exploratory study. *Information and Software Technology* 159 (2023), 107202. https://doi.org/10.1016/j.infsof.2023.107202

[3] Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, et al. 2023. Gemini: A Family of Highly Capable Multimodal Models. *CoRR* abs/2312.11805 (2023). https://doi.org/10.48550/ARXIV.2312.11805 arXiv:2312.11805

[4] 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 (FASE'08/ETAPS'08)*. Springer-Verlag, Berlin, Heidelberg, 93–96.

[5] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, et al. 2020. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (Eds.). https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html

[6] 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. https://doi.org/10.1109/ICSA59870.2024.00019

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

[8] 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. https://doi.org/10.1109/COMPSAC.2012.36

[9] Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, et al. 2024. Scaling Instruction-Finetuned Language Models. *J. Mach. Learn. Res.* 25 (2024), 70:1–70:53. https://jmlr.org/papers/v25/23-0870.html

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

[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, Jill Burstein, Christy Doran, and Thamar Solorio (Eds.). Association for Computational Linguistics, 4171–4186. https://doi.org/10.18653/V1/N19-1423

[12] 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. https://doi.org/10.1109/ASWEC.2008.4483185

[13] 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. https://doi.org/10.1016/j.infsof.2014.01.008

[14] Hamada Elshaboury, Fulvio Re Cecconi, Enrico De Angelis, Luciano Baresi, and Vincenzo Scotti. 2024. NLP-based Data-Enrichment for Building Management. In *Proceedings of the 2024 European Conference on Computing in Construction (Computing in Construction)*, Vol. 5. European Council on Computing in Construction, Chania, Greece. https://doi.org/10.35490/EC3.2024.248

[15] 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. https://doi.org/10.1109/TSE.2018.2861735

[16] Alessio Ferrari, Sallam Abualhaija, and Chetan Arora. 2024. Model Generation with LLMs: From Requirements to UML Sequence Diagrams. In *2024 IEEE 32nd International Requirements Engineering Conference Workshops (REW)*. 291–300. https://doi.org/10.1109/REW61692.2024.00044

[17] Raffaello Fornasiere, Nicolò Brunello, Vincenzo Scotti, and Mark Carman. 2024. Medical Information Extraction with Large Language Models. In *Proceedings of the 7th International Conference on Natural Language and Speech Processing (ICNLSP 2024)*, Mourad Abbas and Abed Alhakim Freihat (Eds.). Association for Computational Linguistics, Trento, 456–466. https://aclanthology.org/2024.icnlsp-1.47/

[18] Dominik Fuchß. 2021. Sketches and Natural Language in Agile Modeling. In *15th European Conference on Software Architecture - Companion (ECSA-C 2021), Virtual online (originally: Växjö, Sweden), September, 13-17, 2021. Ed.: R. Heinrich (CEUR Workshop Proceedings)*, Vol. 2978.

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

[20] Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, et al. 2021. The Pile: An 800GB Dataset of Diverse Text for Language Modeling. *CoRR* abs/2101.00027 (2021). arXiv:2101.00027 https://arxiv.org/abs/2101.00027

[21] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, et al. 2023. Retrieval-Augmented Generation for Large Language Models: A Survey. *CoRR* abs/2312.10997 (2023). https://doi.org/10.48550/ARXIV.2312.10997 arXiv:2312.10997

[22] GitHub. 2021. Introducing GitHub Copilot: your AI pair programmer. https://github.blog/news-insights/product-news/introducing-github-copilot-ai-pair-programmer/

[23] 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. https://doi.org/10.1109/ICSE.2017.9

[24] 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. https://doi.org/10.1145/2379776.2379787

[25] 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. https://doi.org/10.1109/TSE.2006.3

[26] 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). https://doi.org/10.1145/3712003 Just Accepted.

[27] 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. https://doi.org/10.1109/ICSME52107.2021.00008

[28] 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*. https://doi.org/10.5445/IR/1000178589

[29] Tobias Hey, Jan Keim, Anne Koziolek, and Walter F. Tichy. 2020. NoRBERT: Transfer Learning for Requirements Classification. In *28th IEEE International Requirements Engineering Conference, RE 2020, Zurich, Switzerland, August 31 - September 4, 2020*, Travis D. Breaux, Andrea Zisman, Samuel Fricker, and Martin Glinz (Eds.). IEEE, 169–179. https://doi.org/10.1109/RE48521.2020.00028

[30] Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, et al. 2024. Large Language Models for Software Engineering: A Systematic Literature Review. *ACM Trans. Softw. Eng. Methodol.* 33, 8 (2024), 220:1–220:79. https://doi.org/10.1145/3695988

[31] Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, et al. 2023. A Survey on Hallucination in Large Language Models: Principles, Taxonomy, Challenges, and Open Questions. *CoRR* abs/2311.05232 (2023). https://doi.org/10.48550/ARXIV.2311.05232 arXiv:2311.05232

[32] Yue Huang, Lichao Sun, Haoran Wang, Siyuan Wu, Qihui Zhang, et al. 2024. Position: TrustLLM: Trustworthiness in Large Language Models. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net. https://openreview.net/forum?id=bWUU0LwwMp

[33] Aaron Hurst, Adam Lerer, Adam P. Goucher, Adam Perelman, Aditya Ramesh, et al. 2024. GPT-4o System Card. *CoRR* abs/2410.21276 (2024). https://doi.org/10.48550/ARXIV.2410.21276 arXiv:2410.21276

[34] Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, et al. 2023. Survey of Hallucination in Natural Language Generation. *ACM Comput. Surv.* 55, 12 (2023), 248:1–248:38. https://doi.org/10.1145/3571730

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

[36] Matthew Jin, Syed Shahriar, Michele Tufano, Xin Shi, Shuai Lu, et al. 2023. InferFix: End-to-End Program Repair with LLMs. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2023)*. Association for Computing Machinery, New York, NY, USA, 1646–1656. https://doi.org/10.1145/3611643.3613892

[37] Dan Jurafsky and James H. Martin. 2009. *Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition, 2nd Edition.* Prentice Hall, Pearson Education International. https://www.worldcat.org/oclc/315913020

[38] 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. https://doi.org/10.1109/ICSA56044.2023.00021

[39] Jan Keim and Anne Koziolek. 2019. Towards Consistency Checking Between Software Architecture and Informal Documentation. In *IEEE International Conference on Software Architecture Companion, ICSA Companion 2019, Hamburg, Germany, March 25-26, 2019.* IEEE, 250–253. https://doi.org/10.1109/ICSA-C.2019.00052

[40] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large Language Models are Zero-Shot Reasoners. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, et al. (Eds.). http://papers.nips.cc/paper_files/paper/2022/hash/8bb0d291acd4acf06ef112099c16f326-Abstract-Conference.html

[41] 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 (CBSE '15).* Association for Computing Machinery, New York, NY, USA, 21–26. https://doi.org/10.1145/2737166.2737177

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

[43] 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. https://doi.org/10.1109/ICSE43902.2021.00040

[44] 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.

[45] Haoyu Lu, Wen Liu, Bo Zhang, Bingxuan Wang, Kai Dong, et al. 2024. DeepSeek-VL: Towards Real-World Vision-Language Understanding. *CoRR* abs/2403.05525 (2024). https://doi.org/10.48550/ARXIV.2403.05525 arXiv:2403.05525

[46] Patrick Mäder and Alexander Egyed. 2012. Assessing the effect of requirements traceability for software maintenance. In *2012 28th IEEE International Conference on Software Maintenance (ICSM).* 171–180. https://doi.org/10.1109/ICSM.2012.6405269

[47] Patrick Mäder and Alexander Egyed. 2015. Do developers benefit from requirements traceability when evolving and maintaining a software system? *Empirical Software Engineering* 20 (2015), 413–441. https://doi.org/10.1007/s10664-014-9314-z

[48] Johannes Meier, Heiko Klare, Christian Tunjic, Colin Atkinson, Erik Burger, et al. 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. https://doi.org/10.5220/0007396401190130

[49] Meta AI. 2024. Llama 3.2: Revolutionizing Edge AI and Vision with Open, Customizable Models. https://ai.meta.com/blog/llama-3-2-connect-2024-vision-edge-mobile-devices/

[50] C. Mills, J. Escobar-Avila, and S. Haiduc. 2018. Automatic Traceability Maintenance via Machine Learning Classification. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME).* 369–380. https://doi.org/10.1109/ICSME.2018.00045

[51] Shervin Minaee, Tomás Mikolov, Narjes Nikzad, Meysam Chenaghlu, Richard Socher, et al. 2024. Large Language Models: A Survey. *CoRR* abs/2402.06196 (2024). https://doi.org/10.48550/ARXIV.2402.06196 arXiv:2402.06196

[52] Elliot Nelson, Georgios Kollias, Payel Das, Subhajit Chaudhury, and Soham Dan. 2024. Needle in the Haystack for Memory Based Large Language Models. *CoRR* abs/2407.01437 (2024). https://doi.org/10.48550/ARXIV.2407.01437 arXiv:2407.01437

[53] OpenAI. 2023. GPT-4 Technical Report. *CoRR* abs/2303.08774 (2023). https://doi.org/10.48550/ARXIV.2303.08774 arXiv:2303.08774

[54] A. Panichella, C. McMillan, E. Moritz, D. Palmieri, R. Oliveto, et al. 2013. When and How Using Structural Information to Improve IR-Based Traceability Recovery. In *2013 17th European Conference on Software Maintenance and Reengineering.* 199–208. https://doi.org/10.1109/CSMR.2013.29

[55] Mauro Pezzè, Matteo Ciniselli, Luca Di Grazia, Niccolò Puccinelli, and Ketai Qiu. 2024. The Trailer of the ACM 2030 Roadmap for Software Engineering. *ACM SIGSOFT Softw. Eng. Notes* 49, 4 (2024), 31–40. https://doi.org/10.1145/3696117.3696126

[56] Denys Poshyvanyk, Malcom Gethers, and Andrian Marcus. 2013. Concept location using formal concept analysis and information retrieval. *ACM Trans. Softw. Eng. Methodol.* 21, 4, Article 23 (2013). https://doi.org/10.1145/2377656.2377660

[57] 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 (ASE '14).* Association for Computing Machinery, 851–854. https://doi.org/10.1145/2642937.2648624

[58] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving Language Understanding by Generative Pre-Training. https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf

[59] 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 (MSR '18).* Association for Computing Machinery, New York, NY, USA, 442–453. https://doi.org/10.1145/3196398.3196415

[60] Machel Reid, Nikolay Savinov, Denis Teplyashin, Dmitry Lepikhin, Timothy P. Lillicrap, et al. 2024. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *CoRR* abs/2403.05530 (2024). https://doi.org/10.48550/ARXIV.2403.05530 arXiv:2403.05530

[61] 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 International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. 83–84. https://doi.org/10.1109/MODELS-C59198.2023.00026

[62] 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. https://doi.org/10.1109/REW57809.2023.00087

[63] D. V. Rodriguez and D. L. Carver. 2020. Multi-Objective Information Retrieval-Based NSGA-II Optimization for Requirements Traceability Recovery. In *2020 IEEE EIT*. 271–280. https://doi.org/10.1109/EIT48999.2020.9208233 ISSN: 2154-0373.

[64] Victor Sanh, Albert Webson, Colin Raffel, Stephen H. Bach, Lintang Sutawika, et al. 2022. Multitask Prompted Training Enables Zero-Shot Task Generalization. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net. https://openreview.net/forum?id=9Vrb9D0WI4

[65] 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)*. Association for Computing Machinery, New York, NY, USA, Article 52. https://doi.org/10.1145/3241403.3241457

[66] Max Schäfer, Sarah Nadi, Aryaz Eghbali, and Frank Tip. 2024. An Empirical Evaluation of Using Large Language Models for Automated Unit Test Generation. *IEEE Transactions on Software Engineering* 50, 1 (2024), 85–105. https://doi.org/10.1109/TSE.2023.3334955

[67] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, et al. (Eds.). http://papers.nips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html

[68] 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).

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

[70] 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. https://doi.org/10.1109/SCAM51674.2020.00012

[71] Alberto Tagliaferro, Simone Corbo, and Bruno Guindani. 2025. Leveraging LLMs to Automate Software Architecture Design from Informal Specifications. In *22nd IEEE International Conference on Software Architecture, ICSA 2025 - Companion, Odense, Denmark, June 31-April 4, 2025*. IEEE.

[72] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, et al. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, et al. (Eds.). 5998–6008. https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html

[73] Fengcai Wen, Csaba Nagy, Gabriele Bavota, and Michele Lanza. 2019. A Large-Scale Empirical Study on Code-Comment Inconsistencies. In *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*. 53–64. https://doi.org/10.1109/ICPC.2019.00019

[74] Zhiyu Wu, Xiaokang Chen, Zizheng Pan, Xingchao Liu, Wen Liu, et al. 2024. DeepSeek-VL2: Mixture-of-Experts Vision-Language Models for Advanced Multimodal Understanding. *CoRR* abs/2412.10302 (2024). https://doi.org/10.48550/ARXIV.2412.10302 arXiv:2412.10302

[75] Aidan Z. H. Yang, Claire Le Goues, Ruben Martins, and Vincent Hellendoorn. 2024. Large Language Models for Test-Free Fault Localization. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering (ICSE '24)*. Association for Computing Machinery, New York, NY, USA, Article 17, 12 pages. https://doi.org/10.1145/3597503.3623342

[76] Peitian Zhang, Ninglu Shao, Zheng Liu, Shitao Xiao, Hongjin Qian, et al. 2024. Extending Llama-3's Context Ten-Fold Overnight. *CoRR* abs/2404.19553 (2024). https://doi.org/10.48550/ARXIV.2404.19553 arXiv:2404.19553