Enroute to Agentic AKM

Rudra Dhar
IIIT Hyderabad
Hyderabad, Telangana, India
rudra.dhar@research.iiit.ac.in

Karthik Vaidhyanathan
IIIT Hyderabad
Hyderabad, India
karthik.vaidhyanathan@iiit.ac.in

Vasudeva Varma
IIIT Hyderabad
Hyderabad, India
vv@iiit.ac.in

Abstract

Architecture Knowledge Management (AKM) is crucial for maintaining current and comprehensive software Architecture Knowledge (AK) in a software project. However AKM is often a laborious process and is not adopted by developers and architects. While LLMs present an opportunity for automation, a naive, single-prompt approach is often ineffective, constrained by context limits and an inability to grasp the distributed nature of architectural knowledge. To address these limitations, we propose an Agentic approach for AKM, where the complex problem of architecture recovery and documentation is decomposed into manageable sub-tasks. Specialized agents for architecture Extraction, Retrieval, Generation, and Validation collaborate in a structured workflow to generate AK. To validate we made an initial instantiation of our approach to generate Architecture Decision Records (ADRs) from code repositories. We validated our approach through a user study with 29 repositories. The results demonstrate that our agentic approach generates better ADRs, and is a promising and practical approach for automating AKM.

Keywords

Agentic AI, LLM, Architecture Decision Record, Architecture Knowledge Management

1 Introduction

Software architecture provides the foundational blueprint of a system, but keeping its documentation accurate and complete remains a challenge. This documentation, which captures key Architectural Knowledge (AK) such as structure, components, and design principles, is vital for long-term maintenance and evolution. Effective Architecture Knowledge Management (AKM) ensures this knowledge is systematically created and preserved, helping teams build, scale, and adapt software efficiently.

Despite its importance, the manual creation and maintenance of architectural documentation remain a significant bottleneck. This is especially true for crucial artifacts like Architecture Decision Records (ADRs), which capture the rationale behind significant design choices. The documentation process is often perceived as a secondary task, leading to records that are incomplete, outdated, or disconnected from the actual implementation. This knowledge gap introduces significant risks, making it difficult for teams to understand the system's design, onboard new developers, and make informed decisions during its evolution.

The recent advancements in Large Language Models (LLMs) present a opportunity to automate AK documentation by analyzing source code repositories, or other relevent documentation. However, a naive single prompt approach, like feeding an entire repository to an LLM is often ineffective and is constrained by limitations like LLM's context window, and results in outputs that are inaccurate, or lacking in essential context.

To overcome these challenges, we propose an agentic approach, a paradigm gaining significant traction for complex tasks [16], including within software engineering [6, 14]. To overcome these challenges, we propose an agentic approach as defined by Sapkota et al. [11]. Frameworks like AutoGen [15] have demonstrated the power of multi-agent collaboration in various domains. We adapt this concept to AKM, introducing an agentic approach where the complex problem of architecture recovery and documentation is decomposed. Our approach is built upon four distinct types of agents, each responsible for a key stage of the process: Architecture Extraction, Retrieval, Generation, and Validation. These specialized agents, each with specific roles and tools, are coordinated by a central orchestrator.

In this paper, we present the design and an instantiation of this agentic approach. In this instantiation our approach analyzes a code repository and produce a set of ADRs. We validate our approach through a user study comparing our system against a baseline single LLM call. The results demonstrate that our agentic approach produces significantly better ADRs establishing it as a promising and practical approach for automating AKM.

The rest of the paper is structured as followes. Section 2 gives an motivation and overview of the Agentic approach, whereas section 3 explains the various agents. Section 4 details our experimentation. Finally section 5 discusses some related works, and section 6 concludes the work.

2 Motivation and Overview

The recent advances in LLMs offer a promising opportunity to automate AKM. By reasoning over code, configuration files, and existing documentation, LLMs can generate consistent and context-aware artifacts, reducing the manual workload for architects. However, applying LLMs to this task presents key challenges. AK is highly distributed and contextual, spanning multiple abstraction levels that exceed a single LLM prompt's capacity. Limited context windows further constrain analysis of large codebases, leading to partial understanding.

Without a structured process, LLM generated outputs often appear plausible but lack depth, omitting design rationales and historical context. Their opaque reasoning also hinders verification. For example, generating ADRs from a repository requires synthesizing information from code, commits, and issue trackers—something a single LLM call typically fails to achieve, resulting in incomplete or misleading documentation.

To overcome these challenges, we propose an agentic approach. Instead of relying on a single model, we employ a multi-agent

Conference'17, Washington, DC, USA
2025. ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

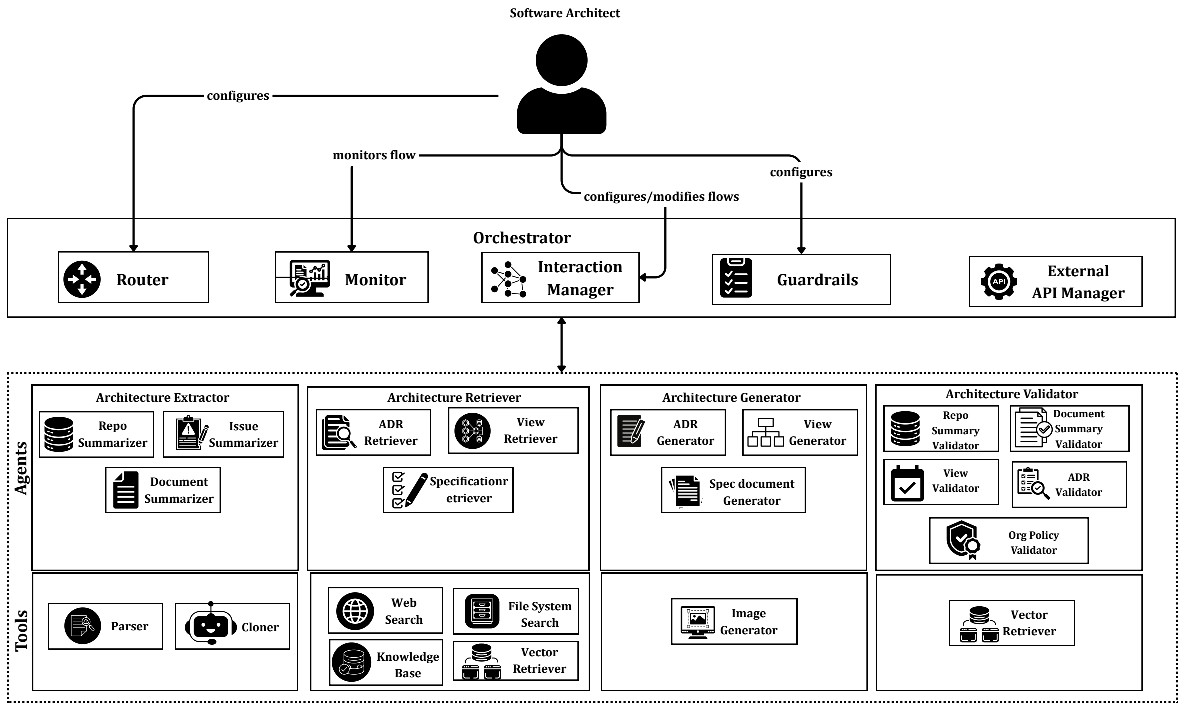

Figure 1: Agentic Approach

system where the complex problem is decomposed into distinct, manageable sub-tasks. Each task is handled by a specialized AI agent with a specific role and toolset, all coordinated by a central orchestrator as shown in Figure 1. This paradigm offers several key benefits:

Decomposition: It breaks down the monumental task of documenting an entire system into smaller, logical steps (e.g., summarize code, retrieve existing docs, generate new ADRs, validate output).
Specialization: It assigns each step to a specialized agent (e.g., a Repo Summarizer, an ADR Generator) that is optimized for that specific task.
Collaboration: Agents pass information and artifacts to one another, ensuring that each stage builds upon a validated foundation. For instance, the summary from the Architecture Extractor provides crucial context for the Architecture Generator.
Validation and Refinement: It incorporates dedicated validator agents that act as quality control checkpoints. This allows for an iterative refinement process, significantly improving the accuracy and reliability of the final output.

3 AKM Agents

Our proposed approach is a multi-agent framework coordinated by a central Orchestrator and supervised by a human Architect as shown in Figure 1. The Architect configures the initial workflow, monitors its execution, and can intervene to modify the flow if necessary. The Orchestrator manages the interaction between four specialized agent groups: Architecture Extractor, Retriever, Generator and Validator. In the rest of the section, we describe some of the Agents. It must be noted, this paper does not provide an exhaustive list of all potential agents within an Agentic AKM. The agents presented constitute an initial configuration that can be expanded and refined in subsequent research.

3.1 Architecture Extractor Agents

This group is responsible for parsing the code repository or other documentation, and creating high-level summaries that serve as the foundational architectural information for all other agents. Some of the agents can be:
Repo Summarizer: This agent analyzes the codebase to understand its overall architecture, primary components, and key functionalities. It uses an LLM to synthesize this analysis into a concise summary.
Issue Summarizer: This agent focuses on the project's issue tracker (e.g., GitHub Issues, Jira). It analyzes bug reports, feature requests, and developer discussions to extract architectural context and pain points, which can inform the AK generation process.
Document Summarizer: This agent ingests existing documentation within the repository (e.g. requirement docs) and uses an LLM to produce condensed summaries.

3.2 Architecture Retriever Agents

This group is responsible for locating and retrieving information from the project's internal knowledge base or external sources. Its retriever agents use various tools—such as web search, vector retrieval, and knowledge base queries. Some of the agents can be:

ADR Retriever: This agent specializes in searching for existing ADRs. It utilizes a Vector Retrieval tool to perform semantic searches over a vector database of past decisions, helping in making and documenting future Design Decisions.
Architecture Diagram Retriever: This agent searches or retrieves existing architectural diagrams within the project's documentation or file system. It may also use image vector retrieval.
Requirement Docs Retriever: This agent uses file System Search and Vector Retrieval to locate and pull information from requirements documents, ensuring that generated architectural decisions align with specified project goals.

3.3 Architecture Generator Agents

This group is responsible for creating new architectural artifacts based on the context provided by the Extractor and Retriever agents. Some of the agents can be:
ADR Generator: This uses an LLM to draft new ADRs based on the repository summary and retrieved information. Each ADR is structured with standard sections like Title, Context, Decision, and Consequences.
Diagram Generator: This agent creates new architectural diagrams (e.g., UML, C4 models). It leverages a Image Generator Model to visualize the architecture described in the repository summary or a specific ADR.
Specification Docs Generator: This agent drafts technical specification documents for new components or services, using an LLM to ensure the documentation is detailed, clear, and consistent with the established architecture.

3.4 Architecture Validator Agents

This group acts as a quality assurance layer, scrutinizing the generated artifacts for accuracy, coherence, and compliance with organizational standards. Some of the agents can be:
Repo Summary Validator: This agent cross-references the summary created by the Extractor against the actual source code to ensure its accuracy and completeness. It uses an LLM to perform this comparative analysis.
Document Summary Validator: This agent checks the summaries of existing documents for factual correctness.
ADR Validator: This agent scrutinizes generated ADRs for logical consistency, format correctness, and overall quality. It may use Vector Retrieval to compare the new ADR against existing ones to check for redundancy or contradiction.
Organization Policy Validator: This agent ensures that generated architectural document comply with organizational best practices or predefined architectural principles stored in a knowledge base.

4 Experiments

To test the viability and effectiveness of our proposed approach, we made a instantiation of it with a multi-agent system to automate the creation of ADRs from code repositories by dividing the task among specialized agents. The source code for all the experiments alongside the data used is available on GitHub [1].

[1]https://github.com/sa4s-serc/AgenticAKM

4.1 Agentic ADR generation from repository

The workflow begins with a **Repository Summarizer Agent** analyzing the codebase to create a high-level summary. This summary is then validated by a **Summary Checker Agent**. If the summary is rejected, it is sent back to the Summarizer for refinement in a loop that runs up to three times.

Once the summary is approved, an **ADR Generator Agent** uses it to identify significant architectural decisions and draft corresponding ADRs. These drafts are scrutinized by an **ADR Checker Agent** for correctness and quality. Similar to the summarization step, this triggers a refinement loop with the generator for up to three iterations if the ADRs are rejected. Finally the approved ADRs are saved.

The Orchestrator Orchestrates the whole flow. It Orchestrates the iterative process and call the respective Agents when required. This agentic, iterative approach ensures that each stage builds upon a validated foundation, significantly improving the accuracy and relevance of the automatically generated architectural documentation.

4.2 Evaluation Setup

To evaluate the effectiveness of our proposed approach, we conducted a comparative user study. We designed the experiment to assess the quality of ADRs generated by two approaches across two different LLMs. We compared two primary approaches for ADR generation:
Baseline Approach: This method involved extracting key files and components from a given repository and feeding this condensed information directly to an LLM in a single prompt to generate ADRs.
Agentic Approach: This is the multi-agent system detailed in subsection 4.1, which uses a structured, iterative workflow involving summarizer, generator, and checker agents to produce the final ADRs.

For the underlying LLMs, we selected 'Gemini-2.5-pro' and 'gpt-5', which were the top-ranked models on the LmArena leaderboard [2] at the time of our experiment (October 5th, 2025). This resulted in four distinct experimental configurations:

- Baseline with Gemini
- Baseline with GPT
- Agentic with Gemini
- Agentic with GPT

User Study Protocol:
We performed the following steps in the user study:

- We initiated our user study by distributing a study form to recruit participants, targeting students and professionals with a background in software engineering and familiarity with ADRs.
- We received responses from 13 participants with 0 to 6 years of industry experience. They collectively submitted 29 unique code repositories in which they had direct expertise or had actively contributed. Most projects were implemented in Python, with some in Java and front ends in JavaScript or TypeScript, ranging from 1,000 to 350,000 lines of code.
- For each repository, we generated four distinct sets of ADRs, each corresponding to one of four experimental configurations. To ensure an unbiased evaluation, we employed a blind study

Source	Model	Relevance	Coherence	Completeness	Conciseness	Overall
LLM	GPT-5	3.8	3.8	3.7	3.5	3.3
LLM	Gemini	3.8	3.6	3.0	3.4	3.3
Agent	GPT-5	4.1	**4.3**	**3.9**	3.9	3.8
Agent	Gemini	**4.3**	4.1	3.8	**4.1**	**3.9**

Table 1: User study results comparing Agentic vs. Baseline (LLM) approaches across two models. Scores are averaged over 29 repositories on a 5-point scale.

design, anonymizing the sets ("Set 1" to "Set 4") so participants did not know which configuration produced which output.

- The participants were then asked to evaluate all four anonymized ADR sets for their repositories using a separate, structured feedback form.
- Following this, all responses were aggregated and analyzed to compare the efficacy of the different configurations.

The evaluation consisted of two parts:
Quantitative Ratings: Participants provided a star rating (from 1 to 5) for each set of ADRs based on four criteria: Relevance, Coherence, Completeness, Conciseness, and Overall Quality.
Qualitative Feedback: Participants also provided written comments detailing the strengths and weaknesses of the ADRs generated by each of the four configurations.

4.3 Results

The **quantitative** results in Table 1 show that the agentic framework consistently outperforms the LLM only approach across all evaluation metrics. Agentic approach attained the highest overall quality score of 3.9. In comparison, the LLM-only configurations scored 3.3 overall, with notably lower Completeness ratings, indicating occasional omissions and limited reasoning depth. The agentic approach also maintained strong Conciseness (3.9–4.1) and Relevance (4.3-4.1) scores, reflecting an effective balance between brevity and informational richness.

The **qualitative** feedback from participants further reinforces these findings. While some users acknowledged the LLM-only outputs as "to the point" or praised them for "good coverage of the whole repo," others criticized them as "very wordy" or lacking structure. In contrast, the outputs generated via the agentic approach were praised more with comments as "very structured and clear" and more reflective of "actual architectural reasoning." One participant noted that the agentic ADRs "actually captured different underlying decisions," whereas the LLM-only outputs appeared "very abstract and generic."

The combined quantitative and qualitative results demonstrate that the agentic AI framework markedly enhances the quality of generated ADRs over simple LLM calls. Utilizing multi-agent workflows, it yields documentation that is more complete, concise, and contextually accurate, better capturing architectural decisions. These findings underscore the promise of our agentic approach for improving architecture documentation.

5 Related Works

Recent advances in GenAI have begun reshaping software architecture research. Esposito et al. [5] mapped the emerging use of LLMs in architectural reconstruction, documentation, and decision support, while Ivers et al. [7] examined which architectural activities are realistically automatable, stressing the enduring need for human governance. Empirical studies such as Dhar et al. [3] explored LLMs' ability to generate architectural decisions from context, revealing model and prompt dependent variability. Similarly, Manjula and Dube [10] demonstrated the use of LLMs for creating and interpreting architecture diagrams. Collectively, these efforts show that LLMs can support specific architecture-related tasks but lack integration into continuous, repository-aware workflows.

On the other hand, Agentic AI in being heavily used to tackle software engineering problems, with some research focusing on foundational design patterns for building them [9]. For example, Wadhwa et al. [13] and Bouzenia et al. [1] analyzed distributed AI agents that decompose software development work into collaborative reasoning cycles, exposing both potential and challenges.

Agentic systems have also been applied in software architecture, an intersection broadly explored by Vaidhyanathan et al. [12]. Díaz-Pace et al. [4] proposed ReArch, a reflective LLM-based framework in which autonomous agents explore architectural design alternatives and reason about trade-offs. Similarly, Li et al. [8] introduced MAAD (Multi-Agent Automated Architecture Design), where specialized agents collaborate to synthesize and evaluate new architectures.

While prior research has primarily focused on design synthesis, emphasizing the creation of new architectural solutions, our work instead applies agentic reasoning to AKM automating the extraction, refinement, and documentation of architecture knowledge from existing software systems, and advancing knowledge recovery and preservation rather than design generation.

6 Conclusion and Future Works

This paper presented a novel agentic framework for AKM . By decomposing architecture recovery into specialized Extractor, Retriever, Generator, and Validator agents, the system overcomes the limitations of monolithic LLM approaches. A user study reveals that the approach produces better ADRs form code repositories. The findings validate that this agentic paradigm offers a robust and effective methodology for automating AKM.

Future work will expand the framework to generate additional artifacts, such as C4 diagrams, and enhance human-agent collaboration through an architect-in-the-loop model. Longitudinal industrial studies will assess scalability and long-term impact, while new agents will be developed to process unstructured, multi-modal data, capturing richer contextual design knowledge.

Acknowledgments

Students and professionals working in SERC, IIIT Hyderabad

References

[1] Islem Bouzenia and Michael Pradel. 2025. Understanding Software Engineering Agents: A Study of Thought-Action-Result Trajectories. arXiv:2506.18824 [cs.SE] https://arxiv.org/abs/2506.18824

[2] Wei-Lin Chiang, Lianmin Zheng, Ying Sheng, Anastasios Nikolas Angelopoulos, Tianle Li, Dacheng Li, Hao Zhang, Banghua Zhu, Michael Jordan, Joseph E. Gonzalez, and Ion Stoica. 2024. Chatbot Arena: An Open Platform for Evaluating LLMs by Human Preference. arXiv:2403.04132 [cs.AI] https://arxiv.org/abs/2403.04132

[3] Rudra Dhar, Karthik Vaidhyanathan, and Vasudeva Varma. 2024. Can LLMs Generate Architectural Design Decisions? -An Exploratory Empirical study. arXiv:2403.01709 [cs.SE] https://arxiv.org/abs/2403.01709

[4] J. Andrés Diaz-Pace, Antonela Tommasel, Rafael Capilla, and Yamid E. Ramírez. 2025. Architecture Exploration and Reflection Meet LLM-based Agents. In *2025 IEEE 22nd International Conference on Software Architecture Companion (ICSA-C)*. 1–5. doi:10.1109/ICSA-C65153.2025.00015

[5] Matteo Esposito, Xiaozhou Li, Sergio Moreschini, Noman Ahmad, Tomas Cerny, Karthik Vaidhyanathan, Valentina Lenarduzzi, and Davide Taibi. 2025. Generative AI for Software Architecture. Applications, Challenges, and Future Directions. arXiv:2503.13310 [cs.SE] https://arxiv.org/abs/2503.13310

[6] Junda He, Christoph Treude, and David Lo. 2025. LLM-Based Multi-Agent Systems for Software Engineering: Literature Review, Vision, and the Road Ahead. *ACM Transactions on Software Engineering and Methodology* 34, 5 (2025), 1–30.

[7] James Ivers and Ipek Ozkaya. 2025. Will Generative AI Fill the Automation Gap in Software Architecting?. In *2025 IEEE 22nd International Conference on Software Architecture Companion (ICSA-C)*. 41–45. doi:10.1109/ICSA-C65153.2025.00014

[8] Ruiyin Li, Yiran Zhang, Xiyu Zhou, Peng Liang, Weisong Sun, Jifeng Xuan, Zhi Jin, and Yang Liu. 2025. MAAD: Automate Software Architecture Design through Knowledge-Driven Multi-Agent Collaboration. arXiv:2507.21382 [cs.SE] https://arxiv.org/abs/2507.21382

[9] Yue Liu, Sin Kit Lo, Qinghua Lu, Liming Zhu, Dehai Zhao, Xiwei Xu, Stefan Harrer, and Jon Whittle. 2025. Agent design pattern catalogue: A collection of architectural patterns for foundation model based agents. *Journal of Systems and Software* 220 (2025), 112278.

[10] Nishchai Manjula and Akhilesh Dube. 2024. Harnessing generative AI to create and understand architecture diagrams. *International Journal of Science and Research Archive* 13 (12 2024), 3330–3336. doi:10.30574/ijsra.2024.13.2.2601

[11] Ranjan Sapkota, Konstantinos I Roumeliotis, and Manoj Karkee. 2025. Ai agents vs. agentic ai: A conceptual taxonomy, applications and challenges. *arXiv preprint arXiv:2505.10468* (2025).

[12] Karthik Vaidhyanathan and Henry Muccini. 2025. Software Architecture in the Age of Agentic AI. In *European Conference on Software Architecture*. Springer, 41–49.

[13] Nalin Wadhwa, Atharv Sonwane, Daman Arora, Abhav Mehrotra, Saiteja Utpala, Ramakrishna B Bairi, Aditya Kanade, and Nagarajan Natarajan. 2024. MASAI: Modular Architecture for Software-engineering AI Agents. In *NeurIPS 2024 Workshop on Open-World Agents*. https://openreview.net/forum?id=NSINt8lLYB

[14] Yanlin Wang, Wanjun Zhong, Yanxian Huang, Ensheng Shi, Min Yang, Jiachi Chen, Hui Li, Yuchi Ma, Qianxiang Wang, and Zibin Zheng. 2025. Agents in software engineering: Survey, landscape, and vision. *Automated Software Engineering* 32, 2 (2025), 1–36.

[15] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. 2023. AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation. arXiv:2308.08155 [cs.AI] https://arxiv.org/abs/2308.08155

[16] Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, et al. 2025. The rise and potential of large language model based agents: A survey. *Science China Information Sciences* 68, 2 (2025), 121101.