SHORT-PAPER

Knowledge-Based Multi-Agent Framework for Automated Software Architecture Design

YIRAN ZHANG, Nanyang Technological University, Singapore City, Singapore

RUIYIN LI, Wuhan University, Wuhan, Hubei, China

PENG LIANG, Wuhan University, Wuhan, Hubei, China

WEISONG SUN, Nanyang Technological University, Singapore City, Singapore

YANG LIU, Nanyang Technological University, Singapore City, Singapore

Open Access Support provided by:

Nanyang Technological University

Wuhan University

Published: 23 June 2025

Citation in BibTeX format

FSE Companion '25: 33rd ACM International Conference on the Foundations of Software Engineering
June 23 - 28, 2025
Trondheim, Norway

Conference Sponsors:
SIGSOFT

Knowledge-Based Multi-Agent Framework for Automated Software Architecture Design

Yiran Zhang
yiran002@e.ntu.edu.sg
Nanyang Technological University
Singapore

Ruiyin Li
ryli_cs@whu.edu.cn
Wuhan University
Wuhan, China

Peng Liang
liangp@whu.edu.cn
Wuhan University
Wuhan, China

Weisong Sun*
weisong.sun@ntu.edu.sg
Nanyang Technological University
Singapore

Yang Liu
yangliu@ntu.edu.sg
Nanyang Technological University
Singapore

ABSTRACT

Architecture design is a critical step in software development. However, creating a high-quality architecture is often costly due to the significant need for human expertise and manual effort. Recently, agents built upon Large Language Models (LLMs) have achieved remarkable success in various software engineering tasks. Despite this progress, the use of agents to automate the architecture design process remains largely unexplored. To address this gap, we envision a Knowledge-based Multi-Agent Architecture Design (MAAD) framework. MAAD uses agents to simulate human roles in the traditional software architecture design process, thereby automating the design process. To empower these agents, MAAD incorporates knowledge extracted from three key sources: 1) existing system designs, 2) authoritative literature, and 3) architecture experts. By envisioning the MAAD framework, we aim to advance the full automation of application-level system development.

CCS CONCEPTS

• **Software and its engineering** → **Designing software**; **Collaboration in software development**.

KEYWORDS

Large Language Model, Multi-Agent System, Software Architecture

ACM Reference Format:
Yiran Zhang, Ruiyin Li, Peng Liang, Weisong Sun, and Yang Liu. 2025. Knowledge-Based Multi-Agent Framework for Automated Software Architecture Design. In *33rd ACM International Conference on the Foundations of Software Engineering (FSE Companion '25), June 23–28, 2025, Trondheim, Norway*. ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3696630.3728493

*Weisong Sun is the corresponding author.

FSE Companion '25, June 23–28, 2025, Trondheim, Norway

ACM ISBN 979-8-4007-1276-0/25/06.
https://doi.org/10.1145/3696630.3728493

1 INTRODUCTION

Software architecture plays a vital role in the software development process. It serves as the blueprint that ensures systems are scalable, maintainable, and aligned with the business goals specified in the Software Requirements Specification (SRS) [2]. Traditionally, the architecture design process relies heavily on human expertise. This manual approach increases costs and makes the process time-consuming and inconsistent [7]. Therefore, automating architectural design is essential for achieving effective and efficient automation of end-to-end application-level software development.

Many attempts have been made to automate architecture design. Recent studies on code generation have begun to incorporate the architectural design process [8–10]. However, these efforts primarily rely on a straightforward zero-shot prompting strategy. As highlighted by Dhar *et al.* [3], such approaches frequently lead large language models (LLMs) to generate hallucinatory content. A promising solution to this issue is the adoption of multi-agent systems, which leverage interactions among multiple LLM-based agents [17] to enhance reliability and creativity [4, 16]. However, whether a multi-agent system can be used to automate the architecture design process remains unexplored.

To fill this gap, in this paper, we present our vision for an automated software architecture design framework, *Multi-Agent Architecture Design (MAAD)*. The overview of MAAD is shown in Figure 1. MAAD involves collaboration among four architecture agents to jointly implement the system architecture design based on the input SRS. Additionally, to support these agents, we propose potential solutions for extracting architectural knowledge from both existing designs and authoritative sources. By addressing key challenges and opportunities in automated architecture design, we aim to advance the automation of end-to-end software development.

2 MAAD FRAMEWORK

In this section, we will detail our vision of MAAD as shown in Figure 1. The four agents, Analyst, Modeler, Designer, and Evaluator, will collaboratively design the system architecture based on the input SRS. First, we will illustrate the collaboration process.

2.1 Collaboration Process

To facilitate understanding, an example of developing an online bookstore is shown in Figure 2. Initially, the Analyst agent analyzes

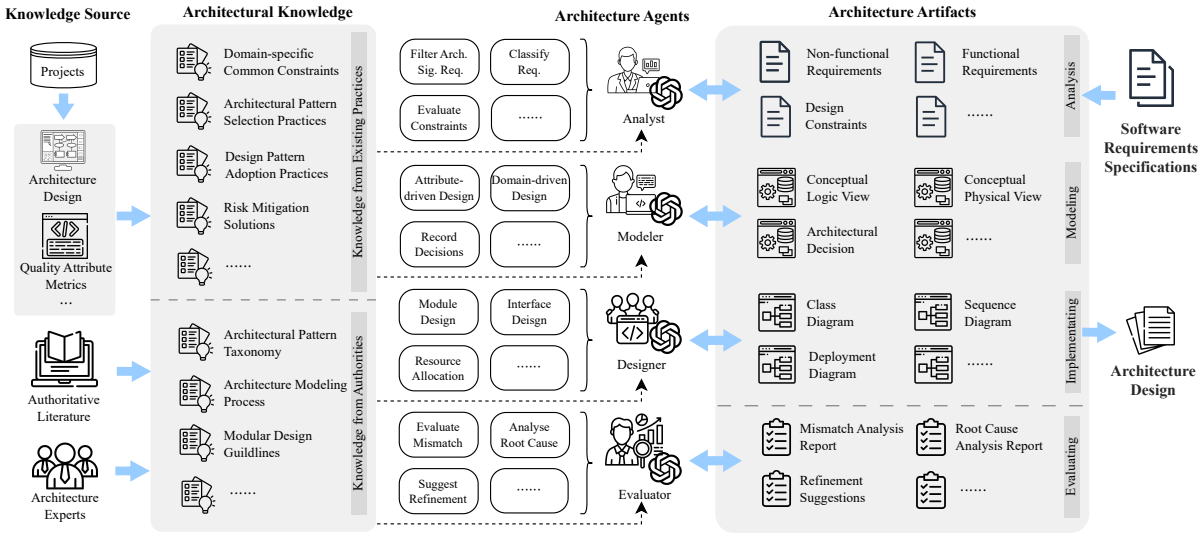

Figure 1: The Overview of Knowledge-Based Multi-Agent Framework for Architecture Design

the input SRS by extracting requirements and constraints that influence the architecture design. Building on the Analyst agent's analysis results, the Modeler agent undertakes two primary tasks: 1) formulating high-level architectural decisions to guide the design, and 2) specifying the domains and prioritized quality attributes that the generated system needs to address. Next, the Designer agent conducts concrete architecture implementation by creating UML diagrams (class diagrams, sequence diagrams, etc.), which serve as blueprints to guide follow-up code development. Finally, the Evaluator agent verifies whether the artifacts align with the input SRS. If any mismatches are found, it identifies the root cause and collaborates with the relevant agent to resolve the issue. The remaining agents then update their respective artifacts accordingly. The architecture design process concludes when the Evaluator confirms the artifacts. The final design, including the designed diagrams, conceptual views, and documented architectural decisions, collectively serve as the foundation for the subsequent code implementation.

2.2 Architecture Agents

2.2.1 Analyst. The Analyst agent is responsible for understanding the SRS, filtering, classifying, and documenting the requirements that impact the architecture. This agent is crucial to ensuring that the subsequent architecture design aligns with business goals.

To accomplish this task, the Analyst agent should possess capabilities to understand the SRS, which includes the following actions: 1) parse and structure the SRS to extract all requirements; 2) filter out architecture-significant requirements (ASRs); 3) classify functional and non-functional requirements, where functional requirements specify the system features and tasks, and non-functional requirements include quality attributes, resource constraints, and other relevant aspects. In addition to these actions, given that the SRS may be imperfect, the Analyst should also 4) identify potential risks in the SRS, such as ambiguous descriptions of system functionalities or conflicting quality attributes, and 5) communicate with stakeholders to address these issues and accordingly refine the SRS.

To support these actions, we propose that the Analyst agent should incorporate the following architectural knowledge: 1) methods for identifying and evaluating architecture-significant requirements, including their criticality and impact on design decisions; 2) understanding of the inherent trade-offs among various quality attributes, such as balancing performance, scalability, and security; 3) expertise in modeling non-functional requirements and mapping quality attributes to appropriate architectural tactics; 4) techniques for identifying and resolving risks in requirements; 5) knowledge of domain-specific constraints that influence architectural decisions.

2.2.2 Modeler. The Modeler agent is to model the overall architecture of the system based on the refined requirements. It will generate 1) architectural decisions such as the selected technology stacks, architectural styles, and patterns; 2) conceptual logical views reflecting the systems' key domains; and 3) conceptual physical views outlining the system's overall deployment topology.

To fulfill these responsibilities, the Modeler should perform the following actions: 1) prioritize the non-functional requirements to facilitate trade-offs among quality attributes; 2) select the appropriate technology stack to meet the requirements; 3) choose suitable architecture styles and patterns to structure the system effectively; 4) identify domain components and their interrelationships to construct conceptual logical views; 5) allocate resources and plan the deployment model to construct conceptual physical views.

The Modeler agent also requires access to specific architectural knowledge to effectively perform its actions: 1) techniques for prioritizing non-functional requirements to balance trade-offs; 2) knowledge of technology stacks, including their capabilities in addressing quality requirements; 3) expertise in architectural styles and patterns, including their applicability and advantages; 4) methods for domain-driven design to identify domain components and establish relationships within the logical views; 5) strategies for resource allocation and deployment planning, including scalability considerations and mapping logical components to physical infrastructure.

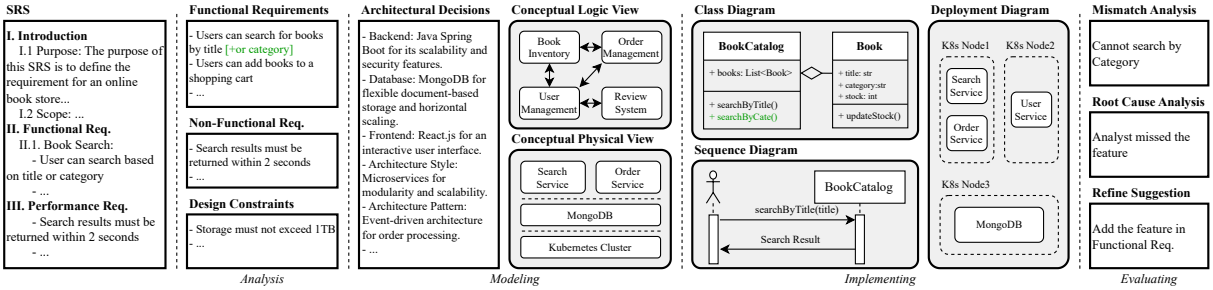

Figure 2: An Example of Proposed Architecture Design Process

2.2.3 Designer. The Designer agent is in charge of refining the conceptual views generated by the Modeler agent into detailed designs that serve as a foundation for subsequent code implementation. Key outputs of the Designer agent include: 1) class diagrams, defining the responsibilities, relationships, and interactions of system modules; 2) sequence diagrams, illustrating the communication and interaction flows between modules; and 3) deployment diagrams, specifying the resource allocation and deployment topology.

To fulfill its responsibilities, the Designer agent should execute the following actions: 1) define the responsibilities and boundaries of each module based on the established logical architecture; 2) design detailed interface specifications to facilitate module collaboration; 3) develop interaction models to outline module interaction mechanisms; 4) refine resource allocation plans to meet system constraints and performance goals.

Likewise, the Designer agent also requires specific architectural knowledge to effectively perform its actions: 1) techniques for modular design, adhering to design principles like SOLID principles [13]; 2) standards and best practices for interface design, ensuring compatibility and reusability; 3) methods for creating interaction models, such as sequence diagrams and collaboration diagrams, to represent communication flows; 4) strategies for resource optimization and allocation, particularly in distributed systems.

2.2.4 Evaluator. The Evaluator agent is tasked with rigorously assessing the architectural artifacts generated by other agents to ensure their alignment with the input SRS. Key outputs from the Evaluator agent include: 1) mismatch analysis reports, documenting discrepancies where the architectural outputs fail to meet the requirements specified in the SRS; 2) root cause analysis reports, identifying the underlying causes of the mismatches and 3) refinement suggestions, providing actionable recommendations.

To fulfill these responsibilities, the Evaluator agent should perform the following actions: 1) evaluate the architectural outputs, including class, sequence, and deployment diagrams, to identify mismatches with the functional and non-functional requirements outlined in the SRS; 2) analyze the identified mismatches and trace back to determine their root causes, such as requirement misinterpretations, design errors, or resource misallocations; 3) formulate refinement suggestions to address the root causes and improve alignment with the system's goals and constraints.

The Evaluator agent also needs specific architectural knowledge to perform its actions: 1) techniques for validating architecture against functional and non-functional requirements, such as quality attribute scenarios and performance evaluations; 2) methods for root cause analysis, focusing on tracing mismatches back to requirement ambiguities or design flaws; 3) architectural evaluation standards specific to domains like real-time systems, cloud computing, and etc.

3 KNOWLEDGE EXTRACTION FOR MAAD

This section outlines the systematic extraction of architectural knowledge from three primary sources: existing projects, authoritative publications, and expert architects. We detail the knowledge extraction process for each source and subsequently demonstrate how the acquired knowledge empowers agents.

3.1 Knowledge from Existing Design

To extract knowledge from existing projects, the process begins by gathering a comprehensive dataset of open-source or publicly available projects from various domains. Architecture recovery tools [6, 19] are employed to extract the architectural design of each selected project. Additionally, static or dynamic analysis tools [15] are utilized to evaluate the quality attributes of these projects, such as performance, scalability, and maintainability. By correlating the recovered architectures with the quality metrics, valuable knowledge can be gained about the impact of architectural design. The knowledge can be applied to tasks like design trade-offs, resource allocation, and quality attribute optimization.

The extracted knowledge from existing projects can support the agents in several ways. For the Analyst agent, it could facilitate the identification and filtering of architecturally significant requirements by examining how similar requirements influenced decisions in other systems. The Modeler agent might leverage observations of architectural patterns and styles, along with their potential trade-offs in achieving desired quality attributes. The Designer agent can draw on extracted module designs and interface specifications derived from logical views, which may highlight design principles. The Evaluator agent can use documented mismatches between requirements and implementations, along with their associated resolutions, to inform their assessments of architectural alignment.

3.2 Knowledge from Authoritative Literature

Extracting knowledge from authoritative literature involves a comprehensive analysis of textbooks, academic papers, and industry

standards that focus on architectural principles, patterns, and frameworks. NLP techniques can be applied to parse and extract structured information, such as definitions of architectural styles, trade-offs of quality attributes, and detailed examples of successful patterns. Furthermore, the literature provides frameworks for risk mitigation, modularization, and evaluation, which can be synthesized into actionable guidelines. Generalized methodologies for designing and evaluating architectures, such as ISO/IEC standards [1], can also be extracted and adapted to support the agents.

The extracted knowledge could strengthen the capabilities of the agents in various ways. The Analyst agent benefits from frameworks for prioritizing non-functional requirements and techniques for identifying ASRs. The Modeler agent gains comprehensive descriptions of architectural styles and patterns, including their trade-offs and suitability for diverse contexts. The Designer agent utilizes guidelines for modular design, emphasizing design principles, as well as standards for interface design and communication modeling. The Evaluator agent relies on methodologies for root cause analysis and frameworks to evaluate architecture against quality attributes.

3.3 Knowledge from Architecture Experts

Knowledge extraction from experts involves engaging with seasoned practitioners through interviews, workshops, and surveys. Structured knowledge elicitation techniques, such as the Delphi method [11], can be employed to gather and organize insights. Experts will be queried about specific architectural challenges, emerging trends, and best practices based on their practical experience. Furthermore, recorded sessions will be transcribed and analyzed using NLP tools to identify recurring themes, actionable advice, and domain-specific recommendations. This process effectively captures the tacit knowledge and practical trade-offs that are often elusive in both literature and project data.

Expert insights significantly enhance the capabilities of the agents. Specifically, the Analyst agent relies on expert advice to elicit and resolve ambiguities in requirements and address domain-specific challenges. The Modeler agent benefits from tailored input on adapting architectural styles and patterns to unique scenarios. The Designer agent can gain practical feedback on emerging trends in module design, interface design, and resource optimization. The Evaluator agent can use expert strategies to identify subtle risks and refine evaluation processes.

4 CHALLENGES AND OPPORTUNITIES

Despite the rapid growth of AI4SE research and its extensive adoption, several challenges persist, presenting valuable avenues for future exploration. Based on our envisioned MAAD framework, we retrospectively and prospectively discuss those challenges and opportunities for future research.

Explainability and Trustworthiness: Many previous studies [14, 18] convey the concerns on the trust on Artificial Intelligence Generated Content (AIGC). Architectural decisions generated by LLM-based agents can be opaque, making it critical to understand how and why an agent arrived at a particular design solution. A key strategy to improve reliability is effective knowledge injection into LLM-based multi-agent frameworks. Retrieval-Augmented Generation (RAG) improves decision-making by grounding agents in external knowledge, such as case studies and authoritative literature [5]. Beyond retrieval, fine-tuning on collected architectural data can also enable agents to internalize best practices and industry standards [12]. Exploring how to effectively utilize or combine these strategies for architectural decision-making is a promising direction for future research.

Coordination and Communication: Each agent role in the MAAD framework could encompass multiple LLMs-based agents to execute input tasks coordinately. The success of any multi-agent system hinges upon the seamless interplay of its constituent agents. This necessitates the design of robust and efficient mechanisms for information sharing, conflict resolution, and decision-making.

- *Information Sharing*: Agents must be able to effectively share relevant information, like their current state, observations, goals, and plans, with other agents. This requires well-defined communication protocols and data formats. Furthermore, mechanisms for filtering and prioritizing information are also crucial to ensure that agents focus on the most relevant data.
- *Conflict Handling*: In multi-agent environments, agents often have competing objectives or disagree on optimal strategies. Effective conflict resolution mechanisms are essential, incorporating negotiation protocols, arbitration processes, or goal prioritization frameworks to systematically address these conflicts.
- *Decision-Making*: Collective decision-making in multi-agent systems requires careful evaluation of individual agents' preferences, priorities, and task requirements. Effective mechanisms may include distributed consensus algorithms or methods for combining individual preferences into coherent collective decisions.

Scalability and Dynamic: In complex architectural tasks, agents should employ dynamic workflows to address evolving requirements. While following structured frameworks, agents can adapt their processes based on problem complexity and constraints. This flexibility allows agents to adapt their workflows dynamically based on the specific demands of the task. Future research could explore adaptive reasoning strategies and dynamic agent configurations where roles evolve with task complexity, enhancing MAAD's adaptability in architectural design.

5 CONCLUSION

In this paper, we envision the Multi-Agent Architecture Design (MAAD) framework, a novel approach to automating software architecture design by leveraging LLM-based agents. By simulating the roles of human architects, MAAD aims to enhance efficiency, scalability, and consistency in architectural tasks. Moreover, the framework integrates knowledge from real-world systems, authoritative sources, and domain experts to guide architectural agents' collaboration in producing high-quality designs based on SRS.

ACKNOWLEDGMENT

This research is supported by National Research Foundation, Prime Minister's Office, Singapore under the Campus for Research Excellence and Technological Enterprise (CREATE) programme; the National Research Foundation, Singapore, and DSO National Laboratories under the AI Singapore Programme (AISG Award No: AISG2-GC-2023-008); the National Natural Science Foundation of China (NSFC) with Grant No. 62402348 and 62172311.

REFERENCES

[1] ISO 42020. 2019. Software, Systems and Enterprise–Architecture Processes. (2019).

[2] Len Bass, Paul Clements, and Rick Kazman. 2012. *Software Architecture in Practice (3rd Edition)* (3rd ed.). Addison-Wesley Professional.

[3] Rudra Dhar, Karthik Vaidhyanathan, and Vasudeva Varma. 2024. Can LLMs Generate Architectural Design Decisions?-An Exploratory Empirical study. In *Proceedings of the 21st IEEE International Conference on Software Architecture (ICSA)*. IEEE, 79–89.

[4] Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. 2024. Improving factuality and reasoning in language models through multiagent debate. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*. OpenReview.net.

[5] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. 2024. Retrieval-Augmented Generation for Large Language Models: A Survey. *arXiv preprint abs/2312.10997v5* (2024).

[6] Joshua Garcia, Daniel Popescu, Chris Mattmann, Nenad Medvidovic, and Yuanfang Cai. 2011. Enhancing architectural recovery using concerns. In *Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 552–555.

[7] David Garlan, Robert Allen, and John Ockerbloom. 2009. Architectural mismatch: Why reuse is still so hard. *IEEE Software* 26, 4 (2009), 66–69.

[8] Sirui Hong, Xiawu Zheng, Jonathan Chen, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, et al. 2023. Metagpt: Meta programming for multi-agent collaborative framework. *arXiv preprint arXiv:2308.00352* (2023).

[9] Md Ashraful Islam, Mohammed Eunus Ali, and Md Rizwan Parvez. 2024. MapCoder: Multi-Agent Code Generation for Competitive Problem Solving. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)*. ACL, 4912–4944.

[10] Feng Lin, Dong Jae Kim, et al. 2024. When llm-based code generation meets the software development process. *arXiv preprint arXiv:2403.15852* (2024).

[11] Harold A Linstone, Murray Turoff, et al. 1975. *The delphi method*. Addison-Wesley Reading, MA.

[12] Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin A Raffel. 2022. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. *Advances in Neural Information Processing Systems* 35 (2022), 1950–1965.

[13] Robert Cecil Martin. 2003. *Agile Software Development: Principles, Patterns, and Practices*. Prentice Hall PTR.

[14] Xinyue Shen, Zeyuan Chen, Michael Backes, and Yang Zhang. 2023. In chatgpt we trust? Measuring and characterizing the reliability of ChatGPT. *arXiv preprint arXiv:2304.08979* (2023).

[15] SonarSource. 2024. *SonarQube*. https://www.sonarqube.org/

[16] Yashar Talebirad and Amirhossein Nadiri. 2023. Multi-agent collaboration: Harnessing the power of intelligent llm agents. *arXiv preprint arXiv:2306.03314* (2023).

[17] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. 2024. A survey on large language model based autonomous agents. *Frontiers of Computer Science* 18, 6 (2024), 186345.

[18] Ruotong Wang, Ruijia Cheng, Denae Ford, and Thomas Zimmermann. 2024. Investigating and designing for trust in AI-powered code generation tools. In *Proceedings of the 2024 ACM Conference on Fairness, Accountability, and Transparency*. ACM, 1475–1493.

[19] Yiran Zhang, Zhengzi Xu, Chengwei Liu, Hongxu Chen, Jianwen Sun, Dong Qiu, and Yang Liu. 2023. Software architecture recovery with information fusion. In *Proceedings of the 31st ACM Joint European Software Engineering Conference (ESEC) and Symposium on the Foundations of Software Engineering (FSE)*. 1535–1547.

Received 17 January 2025; accepted 13 March 2025