Agentic ADR generation from code repository

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

ICSE AGENT

Keywords

Agentic AI, Architecture Decision Record, Architecture Knowledge Management

ACM Reference Format:
Rudra Dhar, Karthik Vaidhyanathan, and Vasudeva Varma. 2018. Agentic ADR generation from code repository. In . ACM, New York, NY, USA, 3 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Software architecture serves as the foundational blueprint for a system, and effectively managing the knowledge behind its design is crucial for long-term maintenance and evolution. Architecture Knowledge Management (AKM) is the discipline dedicated to this practice. A cornerstone of effective AKM is the use of Architecture Decision Records (ADRs), which are documents that capture significant architectural decisions, their contexts, and their consequences. ADRs provide invaluable insight into the "why" behind the system's design, preserving critical rationale that might otherwise be lost over time.

Despite the established benefits of ADRs, their manual creation is often neglected in practice. Developers may perceive documentation as a secondary task, leading to incomplete or non-existent records of architectural knowledge. This gap can significantly hinder project onboarding, system maintenance, and future development efforts.

With the advent of powerful Large Language Models (LLMs), there is a promising opportunity to automate the generation of ADRs directly from source code repositories. However, a naive approach of feeding an entire repository to an LLM is often ineffective. It suffers from practical limitations such as constrained context windows and a high potential for generating inaccurate or superficial outputs due to the complexity and scale of modern codebases.

To address these challenges, we propose an agentic approach for generating ADRs. In this paper, we present a multi-agent system designed to intelligently analyze a code repository and produce high-quality ADRs. We describe our implementation of this system and compare its performance against a baseline LLM-based approach. To validate our method, we conducted a user study which shows that our agentic system produces more satisfactory results, demonstrating its potential as a valuable tool for automating architecture knowledge management

Conference'17, Washington, DC, USA

ACM ISBN 978-1-4503-XXXX-X/2018/06
https://doi.org/XXXXXXX.XXXXXXX

2 Proposed Approach: A Multi-Agent System for ADR Generation

To automate the creation of Architecture Decision Records (ADRs) from existing codebases, we propose a multi-agent system. This approach divides the complex task of ADR generation into distinct, manageable sub-tasks, each handled by a specialized agent. The architecture, depicted in Figure 1, is composed of five distinct agents coordinated by a central Orchestrator Agent.

2.1 Agent Roles and Responsibilities

Orchestrator Agent: This agent acts as the central coordinator of the entire workflow. It is responsible for initiating the process by cloning the repository, directing the flow of artifacts between the summarizer, generator, and checker agents, and managing the iterative refinement loops based on the checkers' feedback. It concludes the process by saving the final approved ADRs.
Repository Summarizer Agent: The first operational agent in the pipeline. Its primary function is to ingest the local path to the cloned source code from the Orchestrator and perform a comprehensive analysis. It synthesizes this analysis into a concise, high-level summary of the repository's architecture, key components, dependencies, and primary functionalities. This summary serves as the foundational context for the subsequent ADR generation.
Summary Checker Agent: This is the first of two quality assurance agents. It receives the summary from the Orchestrator and evaluates it against the repository. If the summary is deemed insufficient or inaccurate, it signals a rejection to the Orchestrator.
ADR Generator Agent: Upon receiving an approved summary from the Orchestrator, this agent identifies significant architectural decisions implicit in the codebase. It leverages the contextual understanding provided by the summary to draft one or more ADRs. Each ADR is structured to include standard sections such as Title, Context, Decision, and Consequences.
ADR Checker Agent: The final quality assurance agent. It scrutinizes the generated ADRs for correctness against the repo-summmary, format consistency, and overall quality. If an ADR fails this check, it signals a rejection to the Orchestrator.

2.2 Workflow and Iterative Refinement

The strength of this model lies in its iterative refinement loops, which are crucial for enhancing the quality of the final output. The process unfolds as follows:

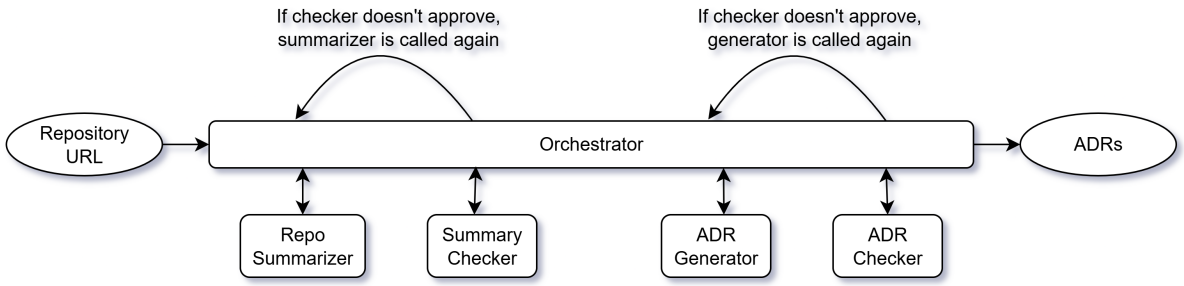

Figure 1: Agentic Approach

Initialization: The process begins when a user provides a repository URL to the Orchestrator, which then clones the repository to a local path.

Summarization and Verification: The Orchestrator invokes the Repo Summarizer. The resulting summary is then passed to the Summary Checker. If the checker rejects the summary, the Orchestrator re-invokes the Repo Summarizer. This refinement loop is repeated a maximum of three times. If the summary is still not approved, the Orchestrator proceeds with the latest available version.

ADR Generation and Verification: Once the summary is approved or the iteration limit is reached, the Orchestrator passes it to the ADR Generator. The drafted ADRs are then forwarded to the ADR Checker. If the checker rejects the ADRs, the Orchestrator re-engages the ADR Generator with feedback. This refinement loop is also repeated a maximum of three times. If the ADRs are not approved after the third attempt, the Orchestrator proceeds with the latest version.

Final Output: Upon approval from the ADR Checker, or after the final iteration, the Orchestrator finalizes the process and outputs the polished, validated ADRs.

This agentic, iterative approach ensures that each stage of the process builds upon a validated foundation, significantly improving the accuracy and relevance of the automatically generated architectural documentation.

3 Experiments

To evaluate the effectiveness of our proposed system, we conducted a comparative user study. We designed the experiment to assess the quality of ADRs generated by two distinct methodologies 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 Section 2, 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 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 recruited participants and asked them to provide code repositories they were familiar with, resulting in a dataset of 20 unique repositories. For each repository, we generated four sets of ADRs, one from each of the four experimental configurations. The participants, who had expert knowledge of their respective repositories, were then asked to evaluate the generated ADRs using a feedback form. The evaluation consisted of two parts:

- **Quantitative Ratings:** Participants provided a star rating (from 1 to 5) for each set of ADRs based on five 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 approaches

4 Retated works

5 Conclusion and Future Works

Acknowledgments

Hiya et al in SA4S.

References

Source	Model	Relevance	Coherence	Completeness	Conciseness	Overall
Agent	GPT-5	4.5	4.0	4.0	4.5	4.0
Agent	Gemini	4.5	4.5	4.0	4.0	4.5
LLM	GPT-5	3.0	3.5	3.0	3.5	3.0
LLM	Gemini	4.0	3.5	2.5	3.5	3.5

Table 1: User study results comparing Agentic vs. Baseline (LLM) approaches across two models. Scores are averaged over 20 repositories on a 5-point scale.