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, 2 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Introduction to software architecture, AKM, ADR. Then why is ADR improtant for AKM.
But people doesn't follow AKM practices and don't write ADRs despite established benifits.
We can give repo to an LLM and get the ADRs. But giving whole repo to LLM doesn't work well, and it will also have context length limitations.
Why not have an agentic way to get ADRs from repo. In this paper we present an approach of how that can be.
We also present an initial agentic implementation and compare it with baseline, which is just calling an LLM.
We validate with a user study, and find Agentic approach works better, and satisfies the user.

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:
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.

Conference'17, Washington, DC, USA

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

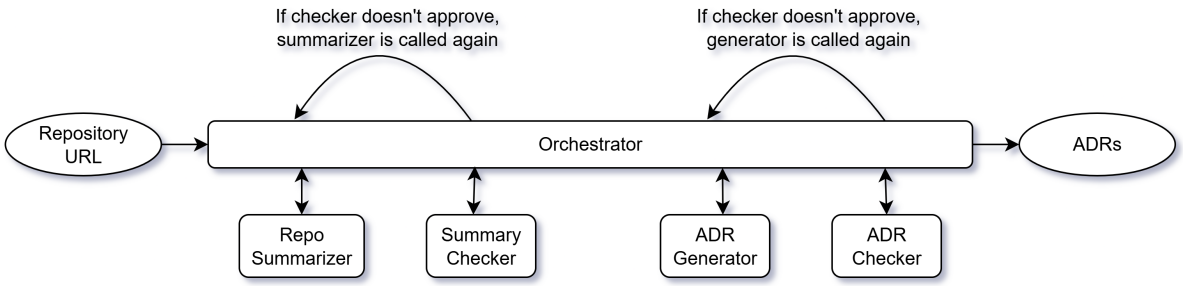

Figure 1: Agentic Approach

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

We used 2 approaches to generate ADRs. Baseline approach, where we get some important part of the repository, and send it to an LLM to generate ADRs. Next is the Agentic approach as explained in the previous section. We used 2 llms, 'Gemini-2.5-pro', and 'gpt-5' which ranks 1nd and 2nd in LmArena at the time of model selection (5th Oct). So performed 4 experiments in total.

We performed an user study to validate our approach. We asked participants to give repositories. We got 20 repositories. We generated ADRs for the given repository with the 4 aforementioned approaches, and gave it back to the participant. The participant then gave us feedback on a form we provided them. They provided feedback for each of the 4 approach. The participant gave a star rating on Relevance Coherence Completeness Conciseness and Overall, for each of the 4. They also gave a qualitative feedback for each.

4 Retated works

Some examples. A paginated journal article [1]

5 Conclusion and Future Works

Acknowledgments

Hiya et al in SA4S.

References

[1] Patricia S. Abril and Robert Plant. 2007. The patent holder's dilemma: Buy, sell, or troll? *Commun. ACM* 50, 1 (Jan. 2007), 36–44. doi:10.1145/1188913.1188915