AI Assistance for Architectural Decision Making: Domain-Driven Context and Prompt Engineering

Olaf Zimmermann[1] and Ronny Seiger[2][0000−0003−1675−2592]

[1] University of St.Gallen, St.Gallen, Switzerland, olaf.zimmermann@unisg.ch
[2] RWTH Aachen University, Aachen, Germany, ronny.seiger@rwth-aachen.de

Abstract. Generative Artificial Intelligence (AI) is receiving a lot of focus in research and practice at present; positions vary from euphoria to "just another tool" to skepticism. Software architecture analysis, synthesis, and evaluation offer many opportunities for AI, e.g., support for knowledge-intensive tasks such as making and recording architectural decisions. It is not yet fully understood how architects can be assisted by AI effectively and efficiently: AI services have to be supervised via input specifications and output validation, requiring architecting skills and application domain expertise. In this paper, we identify architecturally significant requirements to drive the design of a future ecosystem of AI-Assisted Decision Assistance tools (AID). We apply Domain-Driven Design (DDD) to establish a logical component architecture for AID that leverages model-driven knowledge management concepts to accelerate context and prompt/skills engineering and to guard response processing.

Keywords: Architectural Knowledge Management · API Design · Domain-Driven Design · Generative AI · Model-Driven Software Engineering.

1 Introduction and Context

Software architects deal with rich, complex, and ever changing problem and solution spaces (i.e., requirements and design options). In recent years, agile and collaborative modeling, documentation-as-code, and AI services have entered the landscape of methods, notations, and tools. The overall goals of architecting remain: help deliver the desired functionality and quality on time and within budget while keeping the engineering artifacts — from requirement specifications to API contracts, source code, and configuration files — relevant and maintainable. Practicing architects in industry report immense time and delivery pressure; they do not always find enough time to identify the most urgent and important *Architectural Decisions (ADs)* and to evaluate suited design options. Making sound ADs and getting these ADs approved by all involved stakeholders is challenging. Product owners and domain experts specify and then change requirements; enterprise architects establish architectural principles that constrain the design options; developers have preferences regarding concept, technology, and product choices that often differ from those of the architects.

AI promises to boost productivity when performing analysis and design tasks. However, hard evidence for this claim is still missing, at least on the architectural

level. Researchers have demonstrated technical feasibility and initial successes with generative AI in single case studies [1, 6, 8]. Following an empirical design science approach, we investigate the following two related research questions:

(1) How can AI support human architects during architectural analysis, synthesis, and evaluation so that productivity increases and risks decrease?

(2) How should generative and agentic AI be integrated into human activities during the preparation, making, and recording of architectural decisions?

Our response is to combine earlier research results on modeling and managing architectural knowledge, architectural decisions in particular [18], with state-of-the-art concepts and technologies for documentation-as-code and AI usage. LLMs consume text and appreciate text *structure*. Hence, we propose to model project-specific information (e.g., requirements, constraints, ADs already made) in a human- and machine-readable way and inject it into *prompt/skills templates*. The instantiated prompts/skills can then guide and guard the usage of AI services such as chatbots and agents; the human architect stays in the loop but is relieved from tedious prompt/skills writing (we use the two terms prompts and skills interchangeably, assuming text-based LLM input and output).

This short vision paper is structured as follows: Section 2 establishes design objectives and reviews related work. Section 3 proposes a supporting AI Decision Assistance (AID) architecture. Section 4 details the ADR review use case as an example. Section 5 discusses AID and outlines an action plan for future work.

2 AI-Supported AD Making: Status and Requirements

AD Records (ADRs) are typically created and maintained in wiki pages or plain text documents today, following custom templates or publicly available ones such as Nygard's or Markdown ADR (MADR) [5, 15]. Practicing architects often extend the basic Nygard template, for instance with dedicated sections for design alternatives (options). Agile developers use ADRs to drive their *code-level* designs [14]; the ADR log might be their only technical documentation then.

ADRs should discuss the available design options with their pros and cons [12]. Practitioners report that real-world ADRs do not always do so; criteria to compare options often are missing, and consequences sections tend to be incomplete. Open source projects seem to be particularly vulnerable to these problems. A recent review of 300+ ADRs from a publicly available data set [5] confirmed these problems. It also unveiled that key ADs often are not recorded at all; examples include the chosen approach to logical decomposition (clean, onion, or hexagonal style? microservices or modular monolith?), the selected integration style (messaging? RPC? other?) and database paradigm (relational? NoSQL, e.g., key-value pairs?). Such deficiencies limit the value of the ADR logs severely.

AI services seem to be well suited to detect and fix such ADR quality issues, and may also assist with other software architecture tasks. To ground our research in real-world requirements, we asked practicing architects about their position on AI in the context of AD making and capturing, how they use AI today, and which usage scenarios they foresee for the future. Table 1 distills

Table 1. Use of generative/agentic AI by use case and architectural phase

Use case	**Role and value of AI (already practiced, desired)**
Architectural analysis	
Decision driver identification [13]	Suggest decision criteria, find Non-Functional Requirements (NFRs), check completeness and soundness of collected drivers
Decision driver elaboration [13]	Populate Quality Attribute Scenario (QAS) from high-level stakeholder concerns and identified NFRs, give examples
Architectural synthesis (design, documentation)	
AD identification and prioritization [13]	Pick those ADs from decision backlog that matter (e.g., because they are costly to change) and whose time have come
Option exploration [9]	Suggest solutions to problems in a given project context, find comparison criteria, list pros and cons of options
ADR creation and/or elaboration [7, 8, 13, 17]	Turn keywords into full ADR conforming to chosen ADR template (e.g., custom, MADR, Nygard)
Architectural evaluation (review)	
ADR review (linting)	Report room for improvement in ADRs, e.g., regarding consequences balance (good, bad; stakeholder diversity), traces from consequences back to criteria, and template conformance
Review and revision reminders for ADs	Screen ADR log for need to reconsider certain ADs (e.g., due to context changes, technology evolution) to sustain quality
ADR recovery [2]	Document ADs evident in code, track their evolution over time

eight use cases from the responses and our action research, grouped by the three conceptual phases from [11]: *architectural analysis, synthesis* and *evaluation*.

Our practitioner interactions and ideation work also yielded non-functional requirements, including quality goals and constraints: No installation effort should be required, preferably integrating any new capabilities into existing tools to avoid context switches. Ease of use must be taken to the extreme: no documentation should have to be studied to get started, and first successful usage must not take more than a few minutes. ADRs shall be entered as plain text, in IDE/code editors; more rigid input forms and flexible views are welcome too.

Other architecturally significant requirements include content navigability; support for AD(R) classifications and categorizations; knowledge content selectivity and quality; leveraging concepts and content from existing methods (e.g., ADD [4], RCDA); machine readability of knowledge model content; integrability, with CI/CD pipelines in particular, but also with other tools via an open API and/or scriptable Command-Line Interfaces (CLIs); versioning and collaboration via Git; data privacy to protect company-internal information and multi-tenancy capabilities (for Web-based tools); changeability and extensibility to keep up with the fast pace the AI space currently evolves at.

Related work. AI support for architects is a present conference topic. Ivers and Ozkaya [13] list Generative AI (GenAI) competencies and assess the GenAI fit for five common architecture activities. Their 28 sub-tasks overlap with the use cases we found.

Diaz-Pace et al. [9] investigated the option exploration use case, the search for alternatives in particular. Ahmad et al. [1] also focus on architectural analysis, synthesis, and evaluation. The prompts presented in their paper are plain text, applied to a single case. Our vision goes further as we *model* prompt structure and content and generate prompt instances a) from templates capturing community knowledge and b) the project context. Zhou at al. discuss the pros and cons of three prompting strategies, the strengths and limitations of LLM-generated design rationale, and implications for the practical use [17]; they also identify four use cases partially overlapping with the ones we found.

Dhar et al. [7] experimented with few-shot prompting, Retrieval-Augmented Generation (RAG) and LLM fine-tuning to enhance the ability of LLMs to generate ADRs. Cervantes et al. discuss how Attribute-Driven Design (ADD) can be assisted by LLMs [6]; two cases are presented with mixed results reported, including problems regarding chat session memory and context window size.

3 AID: Domain-Driven Component Architecture

To address the requirements from Section 2, we propose to start from our previous work on AD modeling [18] and investigate AI services following *Domain-Driven Design (DDD)* [10]. DDD emphasizes modeling; its strategic patterns express influence relationships and information flows between teams and systems. DDD is well-suited to model complex, rapidly changing domains such as usage of AI in software engineering; the AI services can be seen as one of several *Bounded Contexts* [10] in this setting. Figure 1 sketches existing contexts and context relationships in a *Context Map* [10] that defines the scope of our work.

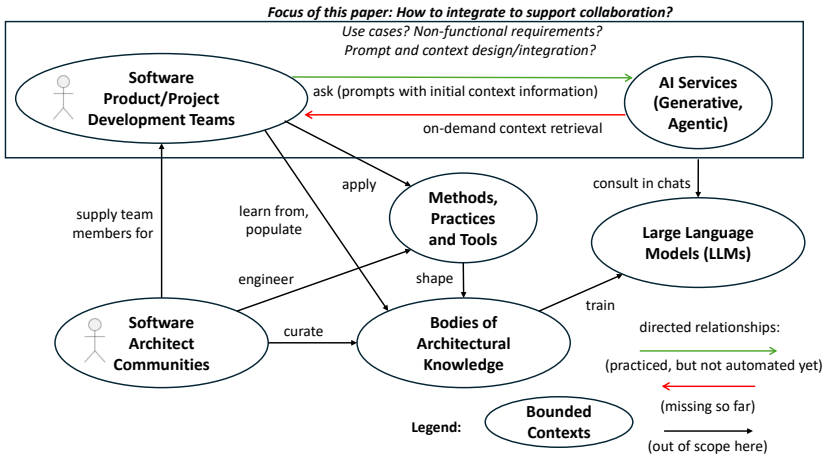

Fig. 1. *Context Map* [10] for AI-assisted architecting: oval nodes represent *Bounded Contexts* [10], directed arrows call out existing or desired context relationships.

Figure 2 zooms into the two contexts at the top of the map. The *Software Product/Project* context on the left side involves both human users (e.g., architects on projects) and other systems (e.g., issue trackers). Application and Domain Services support the use cases from Table 1. The *AD Backlog Aggregate* is a key component that realizes and refines the backlog concept from [11].

The *AI Services* context on the right side generalizes AI services such as ChatGPT, Claude, and Gemini. The Software Product/Project context interacts with the AI Services context in two ways. It prompts the AI — but can also be consulted by the AI on demand via its MCP Server and Open API component, which exposes a model subset in a controlled way. Its provided API and its consumed API share the same data model, a *Published Language* in DDD terms.

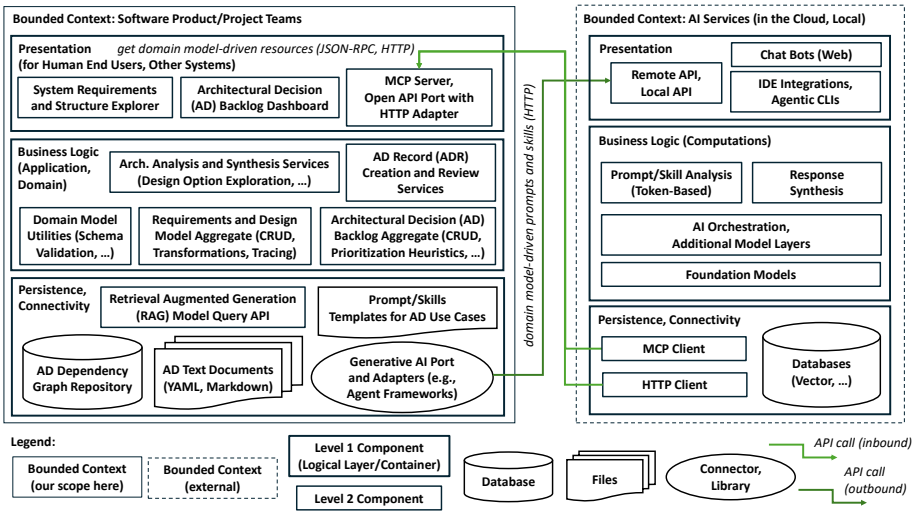

Fig. 2. Use case-driven component architecture: the two mutually linked bounded contexts share a common *Published Language* [10] (i.e., chat/conversation vocabulary).

The Generative AI Port and Adapters decouple the Software Product/Project context from any specific AI models and APIs, supporting changeability, as required to keep up with the expected pace of LLM innovations. This component not only serves as a wrapper/proxy, but also is responsible for logging/archiving the AI interactions and validating the responses semi-automatically. Implementations of this component already exist (commercial and open source): DSPy, PydanticAI and LangChain are three of many examples of agent frameworks.

As persistence paradigm, we propose to combine machine-readable Text Documents (YAML, Markdown) with an AD Dependency Graph Repository to craft an architectural knowledge model accelerating and guiding prompt and context engineering. Text documents can be versioned and collaborated upon via Git; the graph repository adds relationship management and analysis capabilities.

The Prompt/Skills Templates include both prompts for chats with AI bots (e.g., ChatGPT) and skill definitions for agentic AI (e.g., Kiro). Domain model content can be injected to make prompts and skills context-sensitive; it is also exposed in the upstream MCP Server and Open API so that the AI Services may obtain additional information while working on requests. This two-way link closes the collaboration loop between the two bounded contexts (Figures 1, 2).

4 Example: AI Support for ADR Quality Review

We now show a prompt supporting the ADR review (linting) use case introduced in Section 2, combining modeled AD knowledge, project information, and prompt engineering patterns [16]. Due to space constraints, we can only present a subset of the prompt *instance* here; prompt *template*, a full instance that includes the injected ADR and and six exemplary LLM responses are available online [19].

```
You are a software architect with 15 years of experience analyzing
and designing application, integration, infrastructure architectures.
Review an Architectural Decision Record (ADR) regarding 5 criteria:

1. Quality of context section: quality attributes mentioned? [...]
2. Solved problem clearly stated [...]
3. Presence of options with pros and cons, as decision criteria [...]
4. Quality of consequences section [...]
5. ADR template conformance [...]

Use your review findings to derive an overall 'good', 'ok', 'poor' score;
explain the reasoning that lead to this score.

Aliases: [...] When I say "decision driver" or "force", I mean "criteria".

The ADR to be reviewed is: see
https://github.com/apache/james-project/blob/master/src/adr/0037-eventbus.md

Important requirements and design principles are:
see JIRA issue <https://issues.apache.org/jira/browse/MAILBOX-364>

Context information is: see project website at <https://james.apache.org/>

Response format: YAML, see provided example [...]
Style: technical, medium level of detail

Success criteria:
- All five review criteria are discussed.
- The YAML response can be parsed and validated by tools.
```

This prompt has been instantiated from project-specific *requirements* and *context* information and a reusable prompt template following recommended prompt engineering practices from gray and academic literature. The template elements *role*, *aliases*, *re-*

sponse format/style information, and *success criteria* apply the following patterns [16]: Template, Persona, Meta Language Creation, Context Control, and Output Formatter.

We validate the responses of multiple LLMs w.r.t. YAML syntax as well as *plausibility*, *relevance* and *actionability* as domain-specific refinements of precision and recall. Early experimentation results have been mostly positive [19]. We are still in the process of evaluating the responses systematically, following the guidelines by Baltes et al. [3].

5 Discussion and Action Plan

Use cases, component architecture and DDD context map jointly address the research/knowledge questions (1) and (2) from Section 1. We do not employ an "AI can do it all" attitude assuming that AI agents can transform real-world product visions into deployable, production-quality code that meets all functional and non-functional requirements tacitly without human oversight. We rather believe that specifications will continue to be required, with their level of abstraction rising further. Manual ad-hoc prompting or skills engineering is inefficient, does not scale, and does not remove the time/availability issues of practicing architects; our domain model-driven approach to context and prompt/skills engineering vision addresses these limitations.

Probabilistic LLM responses are inherently non-deterministic; hallucinations occur. Practicing architects using AI successfully report that they try the same prompts multiple times and give feedback to the AI along the way. Strong result consolidation and validation are required; our approach aligns well with the emerging practitioner practice of "structured output from LLMs" (cf. ThoughtWorks Technology Radar 4/2026); agent frameworks are beginning to support schema validation, retries, and so on.

Generative and agentic AI can be expected to continue to evolve at rather high speed. Our domain model-driven AID architecture isolates the required adjustments in its adapters and libraries and protects its users from these change dynamics.

In our future work, we plan to implement the presented component architecture in Python. We have tentatively made technology ADs for FastAPI, Jinja2, JSON Schema, neo4j graphs and OpenRouter as AI adapter; action research, case studies, and practitioner surveys will be used to harden and evaluate the AID concepts. The prompt/skills template content will be distilled and curated from community input, personal experience, and peer-reviewed literature. Practitioners have begun to share their prompt collections and agent definitions (i.e., skills) via public repositories; ADR Writer skills for Claude Code and an ADR Creator agent for GitHub Co-Pilot qualify as examples.

An article in the gray literature recently stated "At least for now, GenAI won't be replacing architects anytime soon." and "But they will be replaced by architects who know how to use AI better than anyone else."[3] Our research aims to assist with the "how to use" part of the quote. □

Acknowledgments. This work has received funding from the Swiss National Science Foundation under Grant No. 10002384 (*TaSSAreCt* project).

References

1. Ahmad, A., Waseem, M., Liang, P., Fahmideh, M., Aktar, M.S., Mikkonen, T.: Towards human-bot collaborative software architecting with ChatGPT. In: Pro-

[3] `https://www.cloudwaydigital.com/post/can-ai-replace-software-architects-i-put-4-llms-to-the-test`

ceedings of the 27th International Conference on Evaluation and Assessment in Software Engineering. p. 279–285. EASE '23, ACM, New York, NY, USA (2023)
2. Amalfitano, D., De Luca, M., Santilli, T., Pelliccione, P., Fasolino, A.R.: Automated software architecture design recovery from source code using LLMs. In: Software Architecture. pp. 73–89. Springer Nature Switzerland, Cham (2026)
3. Baltes, S., Angermeir, F., Arora, C., et al.: Guidelines for Empirical Studies in Software Engineering involving Large Language Models (2026), https://arxiv.org/abs/2508.15503
4. Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice, Fourth Edition. Addison-Wesley Professional (2021)
5. Buchgeher, G., Schöberl, S., Geist, V., Dorninger, B., Haindl, P., Weinreich, R.: Using architecture decision records in open source projects – an MSR study on GitHub. IEEE Access **11**, 63725 – 63740 (06 2023)
6. Cervantes, H., Kazman, R., Cai, Y.: An LLM-assisted approach to designing software architectures using add (2025), https://arxiv.org/abs/2506.22688
7. Dhar, R., Kakran, A., Karan, A., Vaidhyanathan, K., Varma, V.: DRAFT-ing architectural design decisions using LLMs (2025), https://arxiv.org/abs/2504.08207
8. Dhar, R., Vaidhyanathan, K., Varma, V.: Can LLMs generate architectural design decisions? an exploratory empirical study. In: 2024 IEEE 21st International Conference on Software Architecture (ICSA). pp. 79–89. IEEE (2024)
9. Diaz-Pace, J.A., Tommasel, A., Capilla, R., Ramírez, Y.E.: Architecture exploration and reflection meet LLM-based agents. In: 2025 IEEE 22nd International Conference on Software Architecture Companion (ICSA-C). pp. 1–5 (2025)
10. Evans, E.: Domain-Driven Design: Tackling Complexity in the Heart of Software. Addison-Wesley (2004)
11. Hofmeister, C., Kruchten, P., Nord, R.L., Obbink, H., Ran, A., America, P.: A general model of software architecture design derived from five industrial approaches. Journal of Systems and Software **80**(1), 106–126 (2007)
12. ISO/IEC/IEEE: International Standard for Software, Systems and Enterprise–Architecture Description (ISO/IEC/IEEE 42010:2022(E)) (2022). https://doi.org/10.1109/IEEESTD.2022.9938446
13. Ivers, J., Ozkaya, I.: Will generative AI fill the automation gap in software architecting? In: 2025 IEEE 22nd International Conference on Software Architecture Companion (ICSA-C). pp. 41–45 (2025)
14. Keeling, M.: The psychology of architecture decision records. IEEE Software **39**(6), 114–117 (2022)
15. Kopp, O., Armbruster, A., Zimmermann, O.: Markdown architectural decision records: Format and tool support. In: ZEUS. pp. 55–62 (2018)
16. White, J., Fu, Q., Hays, S., Sandborn, M., Olea, C., Gilbert, H., Elnashar, A., Spencer-Smith, J., Schmidt, D.C.: A prompt pattern catalog to enhance prompt engineering with ChatGPT. In: Proc. of PLoP '23. The Hillside Group, USA (2023)
17. Zhou, X., Li, R., Liang, P., Zhang, B., Shahin, M., Li, Z., Yang, C.: Using LLMs in generating design rationale for software architecture decisions. ACM Trans. Softw. Eng. Methodol. (Dec 2025)
18. Zimmermann, O., Koehler, J., Leymann, F., Polley, R., Schuster, N.: Managing architectural decision models with dependency relations, integrity constraints, and production rules. Journal of Systems and Software **82**(8), 1249–1267 (2009)
19. Zimmermann, O., Seiger, R.: Example ADR review prompt and corresponding LLM-based ADR reviews (Jun 2026). https://doi.org/10.5281/zenodo.20830043