Will Generative AI Fill the Automation Gap in Software Architecting?

James Ivers
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, USA
jivers@sei.cmu.edu

Ipek Ozkaya
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, USA
ozkaya@sei.cmu.edu

***Abstract*—Researchers are aware that software architects lack effective automation to support much of their work. Generative AI (GenAI) is sparking research interest regarding its potential role in filling this gap, inspired by promising applications of GenAI to other software engineering activities. In this paper, we aim to reflect and sharpen this conversation from the vague "how can GenAI be applied to architecture" to "which architecture activities are most amenable to application of GenAI." We stress the importance of considering contributions in the context of workflows and reflect on the alignment (or lack thereof) of GenAI with the nature of common architecture tasks through the discussion of five common architecture activities. We offer guiding criteria to assist architecture researchers in focusing on activities that are both amenable to automation and likely to obtain significant utility from GenAI.**

***Index Terms*—software architecture, generative AI, LLM, architecture decision making, software architecture automation, architecting workflows**

I. Introduction

The rapid evolution of generative AI (GenAI) models, along with the tools and services built on them, is driving renewed excitement in software engineering to improve automation for a wide range of software engineering activities. The same is true for software architecture. Providing architects with effective automation to support architecting activities has been a longstanding challenge, significantly complicated by the abstraction gap between architecture and code [1]. The lack of effective automation can, in fact, be considered among the top challenges in software architecture [2] [3], one that has not received enough research emphasis.

Today's architects have limited, if any, effective automation to use either during architecting or after their initial realization, such as keeping documentation current and monitoring implementations for conformance with the architecture. Instead, they rely on a lot of manual effort and an ad hoc collection of general purpose tools such as IDEs, PowerPoint (the ultimate tool for architects!), spreadsheets, and static analyzers.

GenAI is renewing hopes for better automation for architects, but creating tools that meet an appropriate mix of relevance, reliability, and utility remains a challenge. Architecture knowledge management has already received some early investigation [4]–[6]. However, preliminary findings on the use of GenAI assistants in software engineering suggest that while these tools can help, ideal improvements in productivity and quality are unlikely to stem solely from these tools [7]. Instead, meaningful progress will come from rethinking workflows and integrating expert judgment or non-AI tools with their use of GenAI. The case is no different for architecting.

In this paper, we aim to sharpen research focus from the vague "how can GenAI be applied to architecture" to "which architecture activities are most amenable to application of GenAI" by reflecting on the alignment of GenAI and architecture activities. Where would automation be most beneficial for specific architecting activities? Which architecting activities naturally align with GenAI competencies? How can sufficient, high-quality data be accumulated to improve GenAI's training corpus? What gaps need to be addressed through further software architecture research? The answers to these questions will involve effective and seamless orchestration of automation and human-in-the-loop workflows. This paper, by bringing the focus of attention to the architecting process and its automation challenges, aims to inspire a research roadmap for software architecture that can take advantage of GenAI where it is most appropriate, complementing it with other automation and practices to remove barriers.

The remainder of this paper is structured as follows. Section II presents an overview of competencies and limitations of GenAI from the lens of architecture. Section III discusses how architecting activities align with GenAI. Section IV introduces fruitful directions for research in GenAI and architecting, and Section V lays out future directions and concludes the paper.

II. Unpacking GenAI

GenAI models are deep neural networks trained on vast datasets (books, code, articles, images, websites, etc.) to identify and learn underlying patterns and relationships in their respective domains [8]. GenAI uses a probabilistic and randomized approach to select the 'next token' in its output sequence, which can mimic correctness and fluency for an end-user but can also lead to errors and omissions. Large language models (LLMs), GenAI models trained on textual data, have found application in software engineering due to their natural alignment with text-based inputs such as code [9].

Architecture research has faced several challenges for years: 1) Publicly available documented architectures are scarce,

To appear in 2025 IEEE 22nd International Conference on Software Architecture Companion (ICSA-C), 2025

and examples differ significantly in form and semantics. Generating clean, consistent training data from this input requires significant effort. 2) Publicly available architectures skew towards small examples (e.g., open-source libraries) or highly precedented examples (e.g., web-based interfaces) over industry scale and unprecedented systems. 3) Validating the effectiveness of automation for several architecture activities relies on the subjective opinions of representative stakeholders, which is difficult to reflect in simulated or classroom exercises.

GenAI research for architecture, with its reliance on high-quality training data exacerbates these challenges. This disconnect suggests that architects should moderate their expectations for the utility of today's GenAI for architecting. However, architecture research can still achieve meaningful progress by focusing on alignment of GenAI's competencies and limitations with the sub-tasks of architecting workflows. Tasks that align with limitations, such as analysis or problem solving, suggest a need to focus on compensating approaches that mitigate these limitations, such as static analyses, in-context learning, retrieval-augmentation generation (RAG), and other tools, to make integrated workflows useful in practice.

A. Competencies of GenAI

Brainstorming, summarization, and application of patterns are natural uses of GenAI that are also integral parts of many architecting activities.

Brainstorming is an act of generating ideas or alternatives. When brainstorming, drawing on common wisdom is generally a helpful way to avoid missing ideas, as long as the signal-to-noise ratio is reasonable. GenAI models help brainstorming because they have been trained on data such as common descriptions of systems, technologies, and requirements.

Summarization involves expressing the most important ideas in a clear and concise way. For example, providing a list of transaction types mentioned in a requirements document and classifying requirements into categories (e.g., quality attributes) are two forms of summarization. GenAI works well for summarization because in addition to being trained on very large amounts of data, their models take into account the context of surrounding text.

Applying patterns is conversion of potentially unstructured inputs to well-formed outputs. Several activities apply patterns, such as converting problem domain descriptions to architecturally significant requirements (ASRs) expressed as quality attribute scenarios. Code generation also relies on pattern matching. Theoretically, multi-modal GenAI could generate code to match visual diagrams and vice versa [10], but quality of the output for architecture will depend on contextualization and availability of a higher-quality training corpus.

B. Limitations of GenAI

Many architecting activities involve conceptual reasoning such as applying abstractions in context, comparing and analyzing options, and reasoning about collections of decisions. Successfully performing these activities is highly dependent on system context and requires combinations of analytical reasoning models to draw objective conclusions, subjective assessment, contextualization, high-fidelity representation, and fact checking, none of which are natural uses of GenAI.

Objective analysis applies some kind of analytic model (e.g., queueing theory or dependency analysis) to generate results that are categorically true or false, without room for opinion. Given robust quantitative models, accuracy or correctness of results is achievable and repeatable. Objective analysis is neither a pattern matching activity nor probabilistic, hence a misfit for GenAI.

Subjective assessment is a form of analysis whose focus is generating answers to represent the opinions of subject matter experts (SMEs), such as their priorities. Predictions of a GenAI model might accidentally match a subjective opinion, but there is little reason to believe that this would be consistent. Local training or fine-tuning could help (e.g., getting to know SME personalities based on prior projects), but extrapolation of opinions in one context to another is dubious.

Contextualization is projection of architecture knowledge onto a high-fidelity representation that is appropriate for the given context. For example, using the publish/subscribe pattern is a general decision. Determining which specific communication paths within a system of hundreds of components employ publish/subscribe and which do not involves analysis of a complete communication topology. GenAI does not support such high-fidelity representation.

Fact checking involves reporting objective information about an input or determining whether a statement is objectively true. For example, determining whether a list of API changes accompanying a new version of a library is correct and complete is a fact checking problem. There is an objectively correct answer, based on examination of the old and new versions of that library's API. Even if GenAI is provided with the correct inputs (e.g., for summarization), it will not create the same response every time. The need to fact check the fact check undermines practical value.

Early research findings are aligned with this discussion. For example, Jahic and Sami report from their industry study [11] that ChatGPT 4 mixed high-level and low-level architecture concepts inappropriately and missed the hierarchy and dependencies among components. However, the architects interviewed appreciated the brainstorming help in identifying the requirements and design patterns they had not considered.

III. Architecting Activities and GenAI

In this section, we describe five common architecture activities and their alignment with GenAI. For each, we identify common sub-tasks and challenges to their execution. Note that this discussion is not intended to depict a specific process or flow. It simply exemplifies the work conducted in the course of architecting software to highlight opportunities for automation. Table I presents the sub-tasks and a rough assessment of the GenAI fit for each. The authors independently reviewed each sub-task based on the competencies and limitations summarized in Section II before consolidating their assessments. We erred on the generous side by assuming that suitable inputs

are available to each activity and that end-users have sufficient knowledge and experience to execute the activities effectively. We do anticipate changes to some of these assessments as GenAI evolves. This assessment focuses on approximating the current state, not predicting the future.

TABLE I
COMMON ARCHITECTURE ACTIVITIES

Activity	Sub-tasks	GenAI Fit
Define ASRs	Identify relevant stakeholders	+
	Identify stakeholders' concerns	+
	Generate well-formed ASRs	+
	Assess correctness & relevance of ASRs	−−
	Assess coverage of qualities and stakeholder concerns across ASRs	−
	Prioritize ASRs	−−
Design an architecture	Identify collection of decisions needed	+
	Identify dependencies among decisions	−
	Identify alternatives	+
	Compare alternatives (in general)	−
	Select and refine alternative (e.g., place a decision in system context)	−−
	Assess goodness of alternative in context	−−
	Prototype to support comparison and assessment of alternatives	+
Evaluate an architecture	Enumerate new architecture decisions	+
	Enumerate prior architecture decisions	+
	Align decisions with their placement in the architecture	−−
	Identify any conflicts among decisions	−−
	Assess satisfaction of each ASR	−−
Document an architecture	Enumerate architecture decisions	+
	Decide on best representation for each decision (e.g., diagrams or prose)	+
	Decide on best view(s) for each decision	+
	Generate views and supporting text	−−
	Determine which decisions merit rationale	−−
	Generate rationale	+
Reconstruct an architecture	Extract facts from artifacts	−
	Identify architecture abstractions from facts	+
	Assemble architecture views	−−
	Test correctness of architecture views	−−

A. Defining Architecturally Significant Requirements (ASRs)

Defining ASRs is typically a human-centered activity that involves engaging stakeholders to elicit their needs, understand their priorities, and reconcile their differences [12]. The core automation challenge is that acceptable results depend on the subjective opinions of stakeholders rather than any analytic theory that can be programmed.

Several ASR sub-tasks can exploit competencies of GenAI (noted with + in Table I). Sub-tasks like identifying types of stakeholders or their common concerns align with the brainstorming competency of GenAI. GenAI responses that reflect common answers can provide helpful starting points or missing topics to an architect [11]. While these suggestions may not be correct or precise with respect to organizational specifics, they are easily reviewable by an architect.

Other sub-tasks require representation of stakeholders' subjective and perhaps idiosyncratic opinions (noted with −− in Table I). For example, while a starting point for an ASR might be a latency requirement for generating a report, assessing the correctness of an ASR requires assessing whether a specific quantified goal makes sense for that system (N seconds, minutes, or hours). This work is more reliant on subjective assessment than on summarization of common knowledge available in a training corpus, and its effectiveness is significantly hampered by the limitations of GenAI.

Assessing the quality and coverage of ASRs is a mixed case (noted with − in Table I). An aspect of this sub-task is categorizing ASRs in one or more dimensions (e.g., by quality attribute), which is a reasonable match to GenAI's summarization competency. However, determining whether enough ASRs exist for each category is another task that is more reliant on subjective assessment.

B. Designing an Architecture

To design an architecture, an architect needs to make a collection of decisions that collectively satisfy the ASRs. This technical activity involves refining general decisions (e.g., deploy in the cloud) to system-specific details (e.g., which services will be replicated and what protocols will be used for run-time interactions) and system-specific conclusions (e.g., a good decision for one system can be an awful one for another).

Multiple design sub-tasks can productively exploit the brainstorming potential of GenAI. Responses that provide general advice, including advantages and disadvantages, on common decisions and alternatives are useful starting points. GenAI can also suggest alternatives that are most commonly related to those used in systems with similar ASRs. While comparing alternatives is partially supported by this summarization of alternatives, GenAI lacks true analytic reasoning for detailed comparison beyond what it can summarize from training material (meriting a − in Table I).

Most design sub-tasks rely on contextualization and objective analysis, running into the limitations of GenAI. Contextualization of abstract design alternatives to system specifics relies on high fidelity representations that must remain consistent across multiple decisions, a trait that is not guaranteed by the stochastic nature of GenAI, and objective analyses to reason about system-specific consequences of design decisions. Although GenAI can provide general suggestions based on the "wisdom of the masses" derived from conclusions most widely cited in the training corpus, it is poorly suited to drawing context-specific conclusions.

C. Evaluating an Architecture

Architecture evaluation involves predicting whether a collection of decisions will support the ASRs [13]. Predictions can be qualitative (based on expert opinion) or quantitative

(based on analytic models), the latter of which derive greater benefit from automation. For this paper, we bias discussion towards repeatable forms of evaluation that produce high confidence results.

Evaluation is a detailed technical activity that requires understanding how all decisions are allocated to elements, relations, and properties within architecture views, reasoning over the precise semantics of each decision, and interactions among decisions. As such, most sub-tasks of evaluation rely heavily on contextualization, high fidelity representation, and some form of analytic reasoning, all of which are challenging concepts for GenAI. Mapping architecture decisions to ASRs poses additional challenges because ASRs often lack identifiers of specific interfaces or components, complicating correlation with high fidelity architecture representations.

The most promising evaluation sub-steps for GenAI involve preparation tasks, like enumerating decisions for inclusion in the evaluation. Given availability of sufficiently rich source material (documentation, decision records, or notes from design meetings), summarization helps to quickly identify the presence of design concepts like patterns and tactics and elicit nearby context from the source material. In practice, while new decisions are often recorded, it can be more challenging to identify material that contains prior decisions when evolving an existing system, and effective evaluation needs to consider both sets of decisions together.

D. Documenting an Architecture

Documentation is a necessary evil. Its essence is to concisely capture architecture decisions and rationale with sufficient clarity and detail to support the needs of stakeholders like developers and test engineers [14]. It requires precision and accuracy to convey an architect's intent and ultimately help ensure that the right software is built and deployed.

Use of GenAI is best aligned with the getting started portions of documentation, such as providing textbook like guidance for how to organize documentation, which views and representations to use, and where each decision is best conveyed (e.g., in a diagram or supporting text). Essentially, GenAI can provide tutor-like guidance on how such decisions are typically handled in publicly available examples.

Generating actual views, however, requires high fidelity representation of project-specific decisions, using labels and symbols that convey unambiguous semantics, and maintaining consistency across views and their supporting text. This contextualization is poorly aligned with the predictive nature of GenAI, as is the fact checking required to ensure consistency across generated artifacts.

E. Reconstructing an Architecture

Recovering the architecture of existing software through study of available artifacts is referred to as architecture reconstruction [15]. The essence of this activity is analyzing artifacts (primarily source code) to infer the architecture that they realize, a task that typically requires judgment and expertise to span the abstraction gap between architecture and code [1].

Reconstruction activities range from building a general idea to constructing representations that allow architects to draw precise conclusions about proposed changes. GenAI is more helpful with the former. Summarization can be effective at providing an overview of the concepts found in artifacts like source code. For example, using GenAI to summarize a system's fault management strategy could result in a reasonable list of exceptions that are raised. However, using GenAI to summarize how the MVC pattern is employed could generate a great deal of noise, as terms like model and view can be widely used throughout a codebase with different meanings.

Architecture abstractions can be implemented in many different ways within the same programming language, let alone across programming languages. Some concepts are implemented directly by developers, while others are implemented by libraries and frameworks. This diversity of options hampers the effectiveness of GenAI's ability to recognize and apply patterns. Effectiveness of applying patterns or summarization for such tasks is more often dependent on the choice of names used in project artifacts, with more intentional use of architecture terms and abstractions likely to yield more useful results in practice.

While there is potential for extracting some architecture information from artifacts, assembling that information into a well-structure architecture view is a more challenging task, as described in the discussion on documenting an architecture. Assessing the correctness of generated architecture views is a similarly poor fit for GenAI as it relies on fact checking and subjective assessment of the captured architectural intent.

IV. Discussion

Our analysis of the alignment between architecture activities and GenAI in Section III clearly shows that expecting out-of-the-box GenAI to solve automation challenges in architecting is overly optimistic. However, researchers still have significant opportunities ahead of them. Although some architecture research has focused on automation, there remains significant room to improve architecting workflows with automation. GenAI provides an avenue to accelerate and refocus such research, but the following criteria must guide efforts.

Is it aligned with competencies of GenAI? For example, tasks that benefit from brainstorming and for which general wisdom is available can take good advantage of that strength.

Can we compensate for the limitations of GenAI? For example, GenAI makes mistakes, and some tasks are intolerant of mistakes (e.g., reasoning tasks). What tools or approaches can be paired with GenAI to produce acceptable results?

Can the underlying data problem be feasibly addressed? We will risk overstating the obvious. Architecture lacks curated data with clear, consistent semantics. Researchers must ask whether existing data is adequate or new data can be generated.

Is the juice worth the squeeze? Proposed automation must provide practical advantage over the status quo. Does GenAI reduce total workflow effort (including compensating approaches) or produce better outcomes at acceptable cost?

To go beyond low-hanging fruit applications, like brainstorming (already readily available to architects), researchers should consider the following directions:

1) **Understand architects' needs**. Study the activities of practicing architects, including where they spend their time and which tasks would most benefit from automation. Target opportunities that promise big wins over those promising incremental improvements.
2) **Integrate GenAI with non-GenAI**. Create workflows that use the best approach (GenAI or non-GenAI) for each sub-task. For example, discovering all inter-process communication used within a system is important when reconstructing an architecture. GenAI can help generate an expanded set of concepts and APIs to search for using more reliable search tools like those found in IDEs. The pairing is better than either alone. Pairing of static analysis and LLMs has already begun to show utility for activities like refactoring [16].
3) **Expand the scope of automation**. Automation need not be limited to solving a problem; GenAI can be helpful in providing guidance on *how* to solve a problem. For example, rather than using GenAI to determine whether a project depends on a library like log4j, one could ask GenAI how to make that determination. Results could point to unfamiliar tools, online knowledge repositories, or project artifacts that one could examine for an answer.
4) **Improve the training corpus**. Curation is needed to create a robust corpus that includes diverse examples described in consistent, semantically rich terms. Beyond that, we can all do more. Many published articles provide recommendations or conclusions (e.g., "use microservices!") without caveats. Intentional description of assumptions improves the body of knowledge and generates a more nuanced training corpus that can improve the ability of GenAI to produce contextual results.

V. Future Plans

We are following this same advice in shaping a research agenda in effective automation for architects, including GenAI. Our recent focus has been on helping teams realize architecture-scale changes in code more efficiently. For example, based on interview and survey data to identify tasks that lack automation [3], we shifted focus from implementing refactorings to generation of refactoring plans using a combination of search-based techniques and static analysis [17].

Similarly, we are studying how GenAI can help free projects from dated programming languages (a hard barrier to modernization) in a new project. We are combining the promise of GenAI to translate code in the small with static analyses to preserve architectural connections and generate incremental translation plans for large projects that enable incremental validation.

In this paper, our goal in asking whether GenAI will fill the automation gap in software architecting was to look at the this challenge through a new lens. Our answer is a loud "not by itself, but..." GenAI does provide opportunities to make progress, but only if research focuses on the right problems. We need to keep the big picture in mind and focus on activities that are both amenable to automation and likely to result in significant utility.

Acknowledgment

Copyright 2024 Carnegie Mellon University and IEEE. This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. [DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution. DM24-1692

References

[1] J. Ivers, I. Ozkaya, and R. L. Nord, "Can AI close the design-code abstraction gap?" in *2019 34th IEEE/ACM International Conference on Automated Software Engineering Workshop (ASEW)*, 2019, pp. 122–125.
[2] A. Telea, H. Sassenburg, and L. Voinea, "Visual tools for software architecture understanding: A stakeholder perspective," *IEEE Software*, vol. 27, no. 06, pp. 46–53, Nov. 2010.
[3] J. Ivers, R. L. Nord, I. Ozkaya, C. Seifried, C. S. Timperley, and M. Kessentini, "Industry experiences with large-scale refactoring," in *Proc. of the 30th ESEC/FSE 2022*, A. Roychoudhury, C. Cadar, and M. Kim, Eds. ACM, 2022, pp. 1544–1554.
[4] R. Dhar, K. Vaidhyanathan, and V. Varma, "Leveraging generative AI for architecture knowledge management," in *21st ICSA-C*, 2024.
[5] I. Ozkaya, "Can architecture knowledge guide software development with generative AI?" *IEEE Software*, vol. 40, no. 5, pp. 4–8, 2023.
[6] E. Ntentos, S. J. Warnett, and U. Zdun, "Supporting architectural decision making on training strategies in reinforcement learning architectures," in *21st ICSA*, 2024, pp. 90–100.
[7] C. Bird, D. Ford, T. Zimmermann, N. Forsgren, E. Kalliamvakou, T. Lowdermilk, and I. Gazit, "Taking flight with Copilot: early insights and opportunities of AI-powered pair-programming tools," *ACM Queue*, vol. 20, no. 6, pp. 35–57, 2022.
[8] S. Wolgram, "What is ChatGPT doing ... and why does it work?" 2023.
[9] X. Hou, Y. Zhao, Y. Liu, Z. Yang, K. Wang, L. Li, X. Luo, D. Lo, J. Grundy, and H. Wang, "Large language models for software engineering: A systematic literature review," *ACM Trans. Softw. Eng. Methodol.*, vol. 33, no. 8, Dec. 2024.
[10] H. Koziolek and A. Koziolek, "LLM-based control code generation using image recognition," in *Proceedings of the 1st International Workshop on Large Language Models for Code*, ser. LLM4Code '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 38–45.
[11] J. Jahic´ and A. Sami, "State of practice: LLMs in software engineering and software architecture," in *21st ICSA-C*, 2024, pp. 311–318.
[12] L. Chen, M. Ali Babar, and B. Nuseibeh, "Characterizing architecturally significant requirements," *IEEE Software*, vol. 30, no. 2, 2013.
[13] ISO/IEC/IEEE, *ISO/IEC/IEEE 42030:2019 - Software, Systems and Enterprise - Architecture Evaluation Framework*, Int. Org. for Standardization Std., 2019.
[14] ——, *ISO/IEC/IEEE 42010:2022 - Systems and Software Engineering — Architecture Description*, Int. Org. for Standardization Std., 2022.
[15] S. Ducasse and D. Pollet, "Software architecture reconstruction: A process-oriented taxonomy," *IEEE Transactions on Software Engineering*, vol. 35, no. 4, pp. 573–591, 2009.
[16] D. Pomian, A. Bellur, M. Dilhara, Z. Kurbatova, E. Bogomolov, A. Sokolov, T. Bryksin, and D. Dig, "Em-assist: Safe automated extractmethod refactoring with LLMs," in *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering, FSE 2024, Porto de Galinhas, Brazil, July 15-19, 2024*, M. d'Amorim, Ed. ACM, 2024, pp. 582–586.
[17] J. Ivers, C. Seifried, and I. Ozkaya, "Untangling the knot: Enabling architecture evolution with search-based refactoring," in *19th IEEE International Conference on Software Architecture, ICSA 2022, Honolulu, HI, USA, March 12-15, 2022*. IEEE, 2022, pp. 101–111.