International Journal of Advanced Engineering, Management and Science (IJAEMS)
Peer-Reviewed Journal
ISSN: 2454-1311 | Vol-12, Issue-3; May-Jun, 2026
Journal Home Page: https://ijaems.com/
DOI: https://dx.doi.org/10.22161/ijaems.123.2

Mechanisms for preserving architectural consistency and system knowledge context during the transition to generative AI-driven development

Shaliev Andrii

Senior Delivery Director, Client Partnership and Growth @ Trinetix Inc, Nashville TN, USA

Received: 29 Mar 2026; Received in revised form: 28 Apr 2026; Accepted: 03 May 2026; Available online: 06 May 2026

***Abstract*— *This article examines architectural mechanisms for preserving architectural consistency and system knowledge context during the transition to generative-oriented software development. The study adopts an analytical synthesis of recent empirical and review research, treating generative development as a system-level architectural process rather than a model-centric activity. The analysis builds on recent studies on the use of language models in architecturally significant engineering processes, as well as on approaches to explicit knowledge representation, architectural decision capture, and process-level governance of software development. It is shown that the risks of architectural degradation in generative development are driven not so much by the quality of individual generation outputs as by the absence of mechanisms for maintaining architectural invariants and causal relationships between requirements, decisions, and their implementation. The reviewed empirical evidence suggests that preserving repository-level architectural context improves the functional correctness of automatically generated artifacts; however, this effect does not extend to the level of system decomposition and inter-service interactions. Special attention is given to interpreting architectural consistency as a cross-cutting property of the development process, shaped by the interaction of external knowledge representations, architectural decision capture mechanisms, and managed process control loops. It is shown that none of these mechanisms in isolation ensures stable preservation of architectural integrity in generative development. The article may be of interest to researchers and practitioners in the fields of software architecture, architectural knowledge management, and the industrial application of generative technologies.*

Keywords*— *architectural consistency, system knowledge context, generative development, language models, architectural invariants, architectural memory, software architecture.

I. INTRODUCTION

Amid the digital transformation of software engineering and the widespread implementation of intelligent generative technologies, the processes of designing and developing software systems are undergoing substantial changes. Generative-driven development accelerates the creation of software code, architectural artifacts, and design decisions, yet simultaneously exacerbates the problem of preserving architectural consistency and the system knowledge context during the evolution of software systems [5]. Architecture, serving as the carrier of key design decisions and constraints, becomes vulnerable to fragmentation when language models are utilized as active participants in the engineering process.

Modern software systems are characterized by high distribution, modularity, and prolonged lifecycles, during which architectural decisions are repeatedly revised. In this context, architectural knowledge is typically distributed across various artifacts and lacks a single formalized representation. In the context of generative development, the absence

©2026 The Author(s). Published by Infogain Publication, This work is licensed under a Creative Commons Attribution 4.0 License. http://creativecommons.org/licenses/by/4.0/

of mechanisms to retain this knowledge leads to architectural drift and the loss of causal links between requirements, decisions, and their implementation [2]. Even given the formal correctness of automatically generated solutions, the risk of violating global architectural invariants persists, necessitating the development of specialized architectural mechanisms to preserve system knowledge at the level of the entire development process. At the same time, existing research on generative development typically treats architectural consistency either as a side effect of generation quality or as a consequence of the correct use of individual tools. An engineering approach that treats the preservation of architectural consistency as a systemic property of the development process, requiring the purposeful design of architectural mechanisms external to language models, remains insufficiently developed.

The aim of the present study is to form a system-architectural analytical framework describing mechanisms for the sustainable preservation of architectural consistency and system knowledge context during the transition to generative-driven development of software systems. To achieve this goal, the work addresses the following tasks:

- identify architectural risks arising from the use of generative models in architecturally significant development processes;
- analyze and compare mechanisms for preserving architectural knowledge based on explicit knowledge representation, architectural decision capture, and process control;
- determine the boundaries of applicability for generative approaches in the absence of external architectural memory mechanisms;
- synthesize a holistic system-architectural model for the sustainable use of generative technologies in long-lived software systems.

The scientific novelty of the study lies in interpreting the preservation of architectural consistency not as a property of the model used or the quality of individual generation results, but as a systemic architectural property of the development process, formed through a combination of external knowledge layers, mechanisms for capturing design decisions, and institutionalized control and verification loops. This approach allows generative technologies to be viewed not as an autonomous source of design decisions, but as a component of a managed architectural environment. Unlike existing approaches to architectural governance, which focus primarily on static architectural descriptions or decision regulation, the proposed approach emphasizes the dynamics of reproducing architectural invariants under conditions of automated generation and system evolution.

The research hypothesis is that the sustainable preservation of architectural consistency and system knowledge context during the transition to generative-driven development is determined not by the power or universality of the applied language models, but by the presence of formalized mechanisms for architectural memory, decision traceability, and process control that ensure the reproducibility and non-contradiction of the architecture over time.

The scope of the study is limited to industrial-scale software systems developed and evolving under conditions of collective development, distributed architectures, and the application of generative models to support architectural and design decisions. The work does not consider narrowly specialized tasks of autonomous code generation, experimental prototypes, or scenarios that do not imply the long-term preservation of architectural knowledge.

II. MATERIALS AND METHODS

Materials for the study were selected based on publications in peer-reviewed international scientific journals and the arXiv archive for the period 2024–2026, dedicated to the application of language models and knowledge representation mechanisms in architecturally significant software system development processes. The selection included works containing either empirical results or systematic reviews directly addressing issues of preserving architectural consistency, system knowledge context, and the manageability of architectural decisions when using generative models. The final corpus of sources covers approaches based on knowledge graphs, architectural decision records, design process pipelines, and language model constraint analytics.

©2026 The Author(s). Published by Infogain Publication, This work is licensed under a Creative Commons Attribution 4.0 License. http://creativecommons.org/licenses/by/4.0/

The study by Athale et al. [1] demonstrates the use of knowledge graphs as a repository-level representation of architectural context for code generation while preserving dependencies and structural invariants. A systematization of architectural and process mechanisms for integrating foundation models into software engineering is presented in the review by Banitaan et al. [2]. The work of Dehal et al. [3] shows the bi-directional integration of knowledge graphs and language models as a means of explicitly representing and validating system knowledge. A mechanism for preserving the context of architectural decisions through a combination of precedent extraction and model fine-tuning is proposed in the study by Dhar et al. [4]. Empirical limitations of generating architectural decisions by language models without external memory mechanisms are identified by Dhar et al. [5]. A process approach to preserving architectural consistency during semi-automated architectural design is described by Eisenreich et al. [6]. A systematic review of artificial intelligence application in microservice architecture design and associated context loss risks is presented by Narváez et al. [7]. Fundamental limitations of language models related to the lack of stable memory and the growth of hallucinations are summarized by Peykani et al. [8]. A taxonomy of mechanisms for integrating language models with knowledge bases and graphs is presented by Some et al. [9]. A quantitative assessment of the ability of language models to reproduce architectural decision rationales was performed by Zhou et al. [10].

The study employed the method of analytical comparison, based on a comparative analysis of results from empirical and review works regarding mechanisms for retaining architectural context and ensuring architectural decision consistency when using generative models. Comparability was ensured through the use of quantitative indicators and categorical classifications provided in the primary sources, including indicators of decision rationale recall and precision, types of architectural mechanisms, and identified architectural risk zones. A methodological limitation of the study is the exclusive use of published results without independent experimental reproduction, which defines the analytical nature of the conclusions drawn and the scope of their interpretation.

III. RESULTS

Within the framework of this study, the preservation of repository architectural context is viewed as a factor determining the functional correctness of automatic software code generation, measured by the pass@1 indicator [4]. The pass@1 metric represents the probability that the first generated solution successfully passes all verification tests without requiring additional regeneration attempts, and is commonly used in empirical evaluations of code generation systems to approximate real-world developer usage scenarios, where only a limited number of generation attempts is feasible. Unlike approaches interpreting generation quality primarily through semantic relevance or stylistic compliance, correctness in this study is treated as the ability of generated code to compile, integrate, and successfully pass verification tests within a real repository environment, thereby implicitly reflecting compliance with existing architectural assumptions and constraints [7].

Repository architectural context in this study is defined through a graph representation of code entities and their dependencies, which allows for formalizing the system's architectural invariants. Unlike local context delivery based on individual files or implementation fragments, graph context captures modular boundaries and usage relationships that cannot be inferred from textual proximity [1]. Within this formulation, the graph representation of the repository functions as a mechanism for the explicit fixation of architectural invariants, constraining the space of permissible generations by preserving information about dependencies, responsibility boundaries, and component usage rules that would otherwise remain implicit.

Experimental data obtained on the EvoCodeBench task set demonstrate a consistent difference in pass@1 values between generation modes using context extraction based on repository graph representation and strategies that do not account for architectural context [4]. Within the considered framework, the increase in pass@1 is interpreted as a reduction in the share of generated

©2026 The Author(s). Published by Infogain Publication, This work is licensed under a Creative Commons Attribution 4.0 License. http://creativecommons.org/licenses/by/4.0/

solutions that violate inter-component dependencies, interface contracts, and other structural repository constraints. In this sense, the pass@1 indicator is used as an indirect quantitative characteristic of system knowledge context preservation, since successful test completion requires the reproduction of local logic and previously adopted architectural assumptions fixed at the repository level. This effect indicates that architectural consistency manifests primarily at the level of functional code behavior, whereas textual plausibility is not a determining factor of correctness. Figure 1 records that using a repository graph representation as context provides a sustained advantage in pass@1 compared to local and context-independent generation strategies.

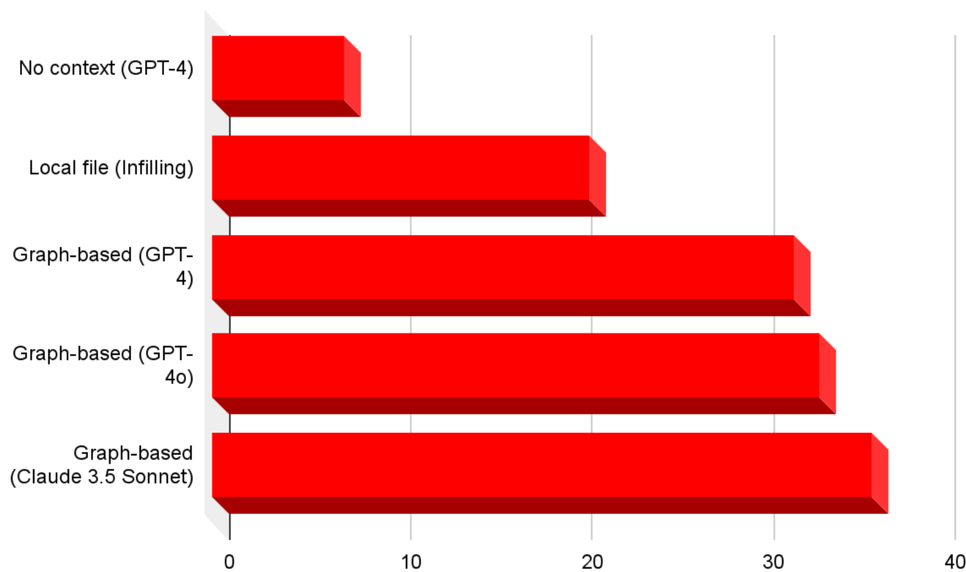

Fig.1 – Effect of preserving repository architectural context on generation correctness (pass@1, EvoCodeBench) (Compiled by the author based on source: [4])

The data presented in the diagram show a monotonic increase in the proportion of correct solutions on the first attempt as the volume and structure of the architectural context increase. In the absence of repository context, generation correctness stands at 7.27%, which fixes the baseline level of functional validity without accounting for system architecture. Using a local file increases the indicator to 20.73%, corresponding to an absolute gain of 13.46 percentage points and an increase of more than 2.8 times relative to the context-independent mode. The transition to a repository graph representation leads to a sharp jump in correctness to 32.00%, meaning an additional gain of 11.27 percentage points compared to local context. Further increase to 33.45% and 36.36% when using different models in graph mode yields a total gain of 29.09 percentage points relative to the baseline scenario, representing a growth of more than 5 times. This dynamic indicates that the decisive factor in increasing correctness is not the change of model, but precisely the preservation of the repository architectural context, which systemically reduces the share of architecturally incorrect generations on the very first attempt.

The analytical aspect is also connected to the fact that graph representation allows implementation elements to be linked with formalized domain and architecture relationships, reducing the probability of hidden violations at modular boundaries [3]. In a process perspective, this effect aligns with the observation that the formalization of architectural context acts as a stabilizing mechanism during automated changes to software systems, preventing the accumulation of architectural drift during their evolution. However, beyond the repository level, this mechanism ceases to be self-sufficient, as architectural decisions related to inter-service interactions and cross-cutting requirements are not anchored in a single formalized context and are distributed among heterogeneous artifacts.

©2026 The Author(s). Published by Infogain Publication, This work is licensed under a Creative Commons Attribution 4.0 License. http://creativecommons.org/licenses/by/4.0/

Transitioning from evaluating generation correctness at the repository level to analyzing system decomposition reveals a consistent vulnerability of architectural consistency in microservice systems, manifesting in specific architectural zones. Architectural decisions formed with automated support demonstrate the greatest instability where explicit fixation of inter-service assumptions and interaction rules is required, which cannot be inferred from local function descriptions [7]. This vulnerability manifests independently of the automation tools used and points to a gap between requirement artifacts and architectural decisions. The recorded gap reflects the absence of a mechanism for transferring and retaining the system knowledge context between requirements, architectural descriptions, and design decisions, resulting in architectural assumptions not being reproduced during automated system decomposition.

Violations are most frequently recorded in the area of data consistency between services, where the architecture requires explicit definition of data ownership and state synchronization rules. In the absence of a formalized representation of these rules, automated solutions reproduce locally correct structures that do not ensure consistent system behavior as a whole. A similar situation is observed in the design of distributed transactions, where compensation mechanisms and failure scenarios remain implicit and, consequently, are lost between architectural artifacts.

A separate group of vulnerabilities is formed by cross-cutting non-functional requirements, including requirements for reliability, stability, and consistent behavior during failures [2]. By their nature, these requirements are distributed across several architectural levels and cannot be correctly recovered from isolated service descriptions. In automated design, such requirements are interpreted fragmentarily, leading to their localization at the level of individual components without accounting for systemic effects. Figure 2 shows the distribution of zones where architectural consistency is most frequently violated during the automated design of microservice systems.

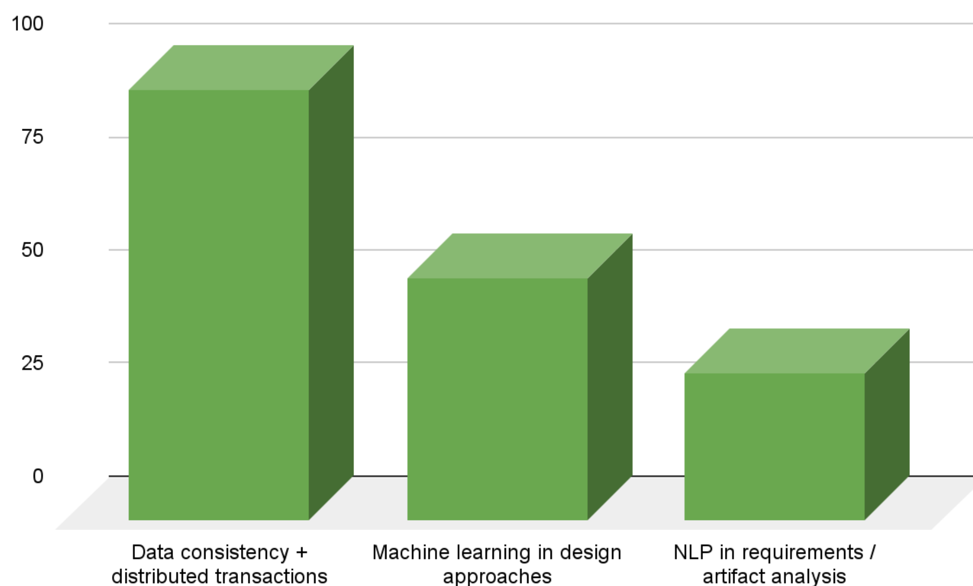

Fig.2 – Where architectural consistency is violated in automated microservices design (Compiled by the author based on source: [7])

The values presented in the diagram fix a pronounced asymmetry in the distribution of architectural problems during the automated design of microservice systems. The highest concentration of violations falls on the zone of data consistency and distributed transactions, which is mentioned in 95.3% of the analyzed studies, indicating its dominant significance as a source of architectural inconsistency. A substantially smaller, but still significant share of works—53.5%—links problems to the application of

©2026 The Author(s). Published by Infogain Publication, This work is licensed under a Creative Commons Attribution 4.0 License. http://creativecommons.org/licenses/by/4.0/

machine learning methods in design processes, reflecting the limitations of automated approaches in interpreting system dependencies. The lowest frequency of mentions is recorded for requirements analysis and architectural artifacts based on natural language processing, amounting to 32.6%, which indicates the secondary nature of this zone compared to problems of consistency and transactional interaction.

The observed concentration of violations in the indicated zones suggests that automated design relies primarily on textual and functional features, failing to ensure the preservation of architectural assumptions over time. These effects are reproduced regardless of specific design methodologies, emphasizing the systemic nature of the problem. The quantitative prevalence of violations in zones of data consistency, distributed transactions, and cross-cutting requirements indicates that automated design tools do not operate with a stable representation of system architectural assumptions but use the fragmented context of individual artifacts.

Thus, the identified vulnerabilities demonstrate that in automated microservice design, architectural consistency is destroyed primarily in zones of inter-service interaction and cross-cutting requirements, pointing to the absence of mechanisms for transferring and retaining system knowledge context between architectural artifacts.

IV. DISCUSSION

Within the framework of this study, the ability of language models to reproduce architectural context is viewed as structurally limited and fundamentally different from the preservation of architectural invariants. The obtained results show that language models are capable of reproducing a significant portion of arguments related to architectural decisions; however, this ability is not accompanied by a mechanism for distinguishing between mandatory and optional elements of decision rationale. Consequently, architectural rationale is recovered as a collection of plausible reasoning rather than a system of strictly fixed constraints.

This property is connected to the nature of language models, which operate on statistical text patterns and do not possess an internal representation of causal architectural dependencies. As a result, arguments that are correct in one architectural context are reproduced in another without verification of their necessity or admissibility. When automatically generated rationales are reused, this leads to the accumulation of reasoning that appears architecturally consistent but does not reflect the mandatory assumptions of the specific system.

Of particular importance is the identified gap between the recall and precision of architectural rationale reproduction. High recall values indicate the ability of language models to cover a wide spectrum of architectural considerations, while low precision values signify the inclusion of arguments that are not architecturally mandatory or are potentially misleading [10]. This effect differs fundamentally from software code generation errors, as it touches upon the level of the system's architectural memory and becomes embedded in decision-making artifacts. Table 1 examines how, despite high reproduction recall, low precision persists, creating a risk of introducing architecturally dangerous arguments.

Table 1 – Characteristics of Design Rationale Generation and Risks of Architectural Inconsistency (Compiled by the author based on source: [10])

Aspect	Quantitative Value	Architectural Interpretation
Precision of generated DR	0.267–0.278	Low selectivity of arguments, risk of rationale dilution
Recall of generated DR	0.627–0.715	Large portion of architectural context is reproduced
Misleading arguments	1.59%–3.24%	Direct threat to architectural invariants
Useful but non-expert arguments	64.45%–69.42%	Context expansion without correctness guarantees

The observed combination of high recall and low precision indicates a fundamental limitation of language models as a means of preserving architectural knowledge. The model reproduces

©2026 The Author(s). Published by Infogain Publication, This work is licensed under a Creative Commons Attribution 4.0 License. http://creativecommons.org/licenses/by/4.0/

decision rationale in the form of coherent text but does not retain the hierarchy of architectural significance of arguments, leading to the gradual dilution of architectural decisions upon reuse [4]. This effect is amplified in the absence of external mechanisms for fixing architectural invariants, such as formalized architectural decision records and structured repositories of architectural knowledge [5].

Consequently, the obtained results indicate that without an external layer of architectural governance, automatically generated decision rationales do not form a stable system architectural memory but become a source of accumulating architecturally heterogeneous reasoning, confirming the limitation of language models in the role of a carrier of long-lived system knowledge.

In this study, preserving architectural consistency during the transition to generative development is viewed as the result of the interaction of several complementary mechanisms, each compensating for specific limitations of language models. Analysis shows that architectural stability cannot be ensured by strengthening a single approach, since context losses occur at different levels of knowledge representation and the engineering process. The considered set of mechanisms can be interpreted as a multi-layered architectural system in which external knowledge representations perform the function of structural constraints, architectural memory performs the function of preserving causal dependencies, and process control performs the function of feedback and deviation correction.

Formalizing architectural knowledge outside the model allows the system structure to be fixed explicitly and limits generation arbitrariness. Using structured knowledge representations ensures the reproduction of entities and links that are not retained in model parameters and are not stably recovered during repeated queries. At the same time, such representations do not preserve the causal logic of architectural decisions and do not contain information about the significance of individual assumptions, limiting their ability to prevent architectural distortions.

Preserving the system's architectural memory through formalized records of decisions and their rationales allows the context of architectural decision-making to be retained over time. This mechanism fixes trade-offs, alternatives, and constraints that are otherwise lost during automated generation. However, analysis results of architectural decision rationales show that language models do not distinguish the mandatory and optional nature of architectural arguments, whereby even formalized architectural memory without additional constraints is subject to gradual dilution.

Process governance of architecture acts as a stabilizing loop, compensating for the lack of self-verification and long-term system context retention capabilities in language models. Managed artifact chains and control cycles allow for identifying violations of architectural assumptions at early stages, but their effectiveness directly depends on the presence of formalized knowledge and architectural memory.

Thus, the discussed results point to the necessity of designing generative-oriented development as a managed architectural system, in which consistency preservation is achieved not by strengthening generative models, but through the composition of architectural mechanisms and processes.

V. CONCLUSION

The transition to generative-oriented software system development should be interpreted not as the introduction of a new class of tools, but as a change in the architectural mode of development, in which system stability is determined by the ability to preserve architectural consistency and system knowledge context over time. Under these conditions, architectural integrity ceases to be a consequence of the quality of individual generation results and is formed as a managed property of the entire engineering process. The conclusions obtained have both theoretical and engineering significance. From a theoretical perspective, the work clarifies the boundaries of applicability for language models as carriers of architectural knowledge. From an engineering perspective, the results form the basis for designing architectural mechanisms that compensate for the limitations of generative technologies in long-lived software systems.

©2026 The Author(s). Published by Infogain Publication, This work is licensed under a Creative Commons Attribution 4.0 License. http://creativecommons.org/licenses/by/4.0/

The main limitation of generative development is connected to the fact that language models reproduce architectural context fragmentarily and do not retain architectural invariants as a system of mandatory constraints. This leads to a shift of architectural risks from the level of local correctness to the level of system evolution, where the dilution of decision rationales, loss of causal links, and violation of inter-artifact consistency accumulate and become a structural problem.

Preserving architectural consistency requires abandoning the view of generative models as autonomous carriers of architectural knowledge. Stability is achieved by externalizing critically significant knowledge beyond the model, fixing architectural decisions and their rationales, and including generative mechanisms in managed design and verification loops. In this configuration, architectural memory, formalized knowledge representations, and process governance act not as auxiliary elements, but as equal architectural layers.

The practical orientation of development must shift from optimizing generation quality to designing architectural composition: generative models as a tool for variability, external knowledge representations as a carrier of structure, architectural memory as a mechanism for decision preservation, and process loops as a means of limiting degradation. It is precisely this composition that allows generative technologies to be used in architecturally significant processes without losing system integrity and makes their application reproducible in long-lived software systems. Thus, the sustainable application of generative technologies in architecturally significant processes requires shifting the focus from optimizing the quality of individual generation results to designing the architectural composition of development. In this composition, language models act as a source of variability, external knowledge representations as a carrier of structural constraints, architectural memory as a mechanism for decision preservation, and process loops as a means of preventing architectural degradation. Such an approach ensures the reproducibility of architectural decisions and makes the use of generative technologies engineering-manageable in long-lived software systems.

REFERENCES

[1] Athale, M., & Vaddina, V. (2025). Knowledge graph based repository-level code generation [Preprint]. arXiv. https://doi.org/10.48550/arXiv.2505.14394

[2] Banitaan, S., Daoud, M., Alquran, H., & Akour, M. (2026). Foundation models in software engineering: A taxonomy, systematic review, and in-depth analysis of testing support. Information, 17(1), 73. https://doi.org/10.3390/info17010073

[3] Dehal, R. S., Sharma, M., & Rajabi, E. (2025). Knowledge graphs and their reciprocal relationship with large language models. Machine Learning and Knowledge Extraction, 7(2), 38. https://doi.org/10.3390/make7020038

[4] Dhar, R., Kakran, A., Karan, A., Vaidhyanathan, K., & Varma, V. (2025). DRAFT-ing architectural design decisions using LLMs. arXiv. https://doi.org/10.48550/arXiv.2504.08207

[5] Dhar, R., Vaidhyanathan, K., & Varma, V. (2024). Can LLMs generate architectural design decisions? – An exploratory empirical study [Preprint]. arXiv. https://doi.org/10.48550/arXiv.2403.01709

[6] Eisenreich, T., Speth, S., & Wagner, S. (2024). From requirements to architecture: An AI-based journey to semi-automatically generate software architectures [Vision paper]. arXiv. https://doi.org/10.48550/arXiv.2401.14079

[7] Narváez, D., Battaglia, N., Fernández, A., & Rossi, G. (2025). Designing microservices using AI: A systematic literature review. Software, 4(1), 6. https://doi.org/10.3390/software4010006

[8] Peykani, P., Ramezanlou, F., Tanasescu, C., & Ghanidel, S. (2025). Large language models: A structured taxonomy and review of challenges, limitations, solutions, and future directions. Applied Sciences, 15(14), 8103. https://doi.org/10.3390/app15148103

[9] Some, L., Yang, W., Bain, M., & Kang, B. (2025). A comprehensive survey on integrating large language models with knowledge-based methods [Preprint]. arXiv. https://doi.org/10.48550/arXiv.2501.13947

[10] Zhou, X., Li, R., Liang, P., Zhang, B., Shahin, M., Li, Z., & Yang, C. (2025). Using LLMs in generating design rationale for software architecture decisions [Preprint]. arXiv. https://doi.org/10.48550/arXiv.2504.20781

©2026 The Author(s). Published by Infogain Publication, This work is licensed under a Creative Commons Attribution 4.0 License. http://creativecommons.org/licenses/by/4.0/