

ArcAssist-Web: Towards a Logical Reference Architecture for GenAI-Assisted Web Architectural Evolution

Vanusa Rodrigues de Oliveira
Federal University of Lavras (UFLA)
Inst. of Sci., Technol. & Innov. (ICTIN)
São Sebastião do Paraíso, Minas Gerais, Brazil
vanusa.oliveira1@estudante.ufla.br

Samira Santos da Silva
Federal University of Lavras (UFLA)
Inst. of Sci., Technol. & Innov. (ICTIN)
São Sebastião do Paraíso, Minas Gerais, Brazil
samirasilva@ufla.br

Bento Rafael Siqueira
Federal University of Lavras (UFLA)
Department of Applied Computing (DAC)
Lavras, Minas Gerais, Brazil
bento.siqueira@ufla.br

Johnatan Alves Oliveira
Federal University of Lavras (UFLA)
Department of Computer Science (DCC)
Lavras, Minas Gerais, Brazil
johnatan.oliveira@ufla.br

ABSTRACT

Web and PWA evolution often requires architectural decisions that cross user-interface, API, data, privacy, and administrative-review boundaries. Generative AI can assist architects in exploring alternatives, provided that its outputs remain grounded in the current Web-system baseline, explicit about assumptions, and subject to human acceptance. This paper presents ArcAssist-Web, a logical reference-architecture proposal for evidence-grounded GenAI assistance in Web architectural evolution. It organizes seven reusable responsibilities: interaction, Web-system and domain context, context and knowledge, prompt orchestration, LLM reasoning, validation and QA, and knowledge management. The proposal is illustrated with an animal-rescue Web/PWA adoption-interest increment. A feasibility-oriented traceability evaluation compares generic prompting with an ArcAssist-Web configuration using frozen Web/domain evidence and a Decision Package contract. Across three retained pairs, both conditions could reach a similar architectural direction, while the configured outputs cited all 15 evidence identifiers and made Web context, assumptions, and validation plans explicit. Because the prompts imposed different output structures, these observations show inspectability and instruction conformance rather than a causal condition effect or architectural correctness.

KEYWORDS

Generative AI, Web systems, software architecture, architectural evolution, architectural decisions, reference architecture

1 INTRODUCTION

Web systems and PWAs evolve through decisions that cross interaction flows, business rules, UI/API/data boundaries, and quality attributes. Even a public form may introduce a domain concept and change browser, API, persistence, and

administrative-review boundaries [9, 13, 19]. Such increments must be related to the current architectural baseline [12].

In concrete terms, ArcAssist-Web takes a change concern and project evidence, requests architectural alternatives and validation work, and returns a Decision Package for architect approval rather than automatic adoption. A reference architecture captures reusable elements, relationships, constraints, and variation for a class of systems [1, 18, 23]; a logical view expresses them as responsibilities and information dependencies while leaving implementation and deployment open [13]. ArcAssist-Web is thus a logical reference architecture for GenAI-assisted Web evolution, not for the target Web applications themselves. Its specialization makes interaction flows, UI/API/data boundaries, data visibility, roles, business rules, and Web quality concerns explicit evidence [9].

For example, given “add adoption interest to a public animal profile,” the configuration supplies the implemented baseline, the browser-to-API flow, public/private data boundaries, planned domain concepts, and unresolved privacy questions. The model returns alternatives and checks linked to that evidence; an architect, not the model, decides whether any proposal becomes retained knowledge.

GenAI can draft alternatives, but without project evidence it may obscure the baseline, constraints, and acceptance responsibility. Accountable integration and expert validation are therefore safeguards [11, 14, 21].

We ask how to organize GenAI assistance so that Web evidence, domain rules, validation, and architect responsibility remain explicit. We contribute: (i) seven jointly used logical responsibilities; (ii) their organization as ArcAssist-Web, including dependencies, constraints, and variation points; and (iii) an animal-rescue Web/PWA scenario in which a visitor submits initial interest from a public animal profile for later staff handling, plus a traceability evaluation of whether decision material remains linked to frozen evidence and exposes assumptions and validation work.

2 BACKGROUND

Software architecture comprises elements, relationships, constraints, and rationale; views make stakeholder concerns and

In: Proceedings of the Brazilian Symposium on Multimedia and the Web (WebMedia'2026). Lavras/MG, Brazil. Porto Alegre: Brazilian Computer Society, 2026.

© 2026 SBC – Brazilian Computing Society.
ISSN 2966-2753

evidence inspectable [8, 13, 19]. For Web systems, evidence spans interfaces, services, domain concepts, APIs, persistence, interaction flows, and quality concerns [9]. A *Web interaction flow* crosses user-facing and administrative activities, interfaces, APIs, and data boundaries; unlike a single user story, it may be informed by several backlog needs [4].

This distinction matters because one apparently local feature can cross a public route, form state, API contract, persistence boundary, and private staff activity while changing which data may be exposed. ArcAssist-Web therefore treats interaction sequence, interface/API/data boundaries, visibility, roles, and Web quality attributes as explicit evidence rather than leaving them implicit in a generic software-change request.

A reference architecture makes shared elements, relationships, constraints, and variation explicit [1, 18, 23]. ArcAssist-Web expresses that knowledge in a logical, technology-independent view of seven responsibilities [13].

Domain rules constrain valid states or transitions; quality constraints preserve attributes such as privacy and modifiability [12]. Because backlogs, ADRs, models, and modules provide partial evidence, assistance should expose omissions, alternatives, consequences, and validation actions [5, 10, 20, 21]. This is salient when a public Web request affects private data, administrative flows, domain rules, or API contracts.

3 RELATED WORK ON GENAI-ASSISTED ARCHITECTING

LLM-system architectures. Reference architectures address LLM data/model handling [3], deployment [15], and accountable agents [14], but not evolution of an existing Web baseline. Architecting automation should preserve human acceptance authority [11].

Generation and validation. Related work orchestrates LLM agents [2], generates architectures or diagrams [6, 7, 22], and motivates validation through conformance or formal analysis [16, 20].

Evidence from WebMedia. At WebMedia, Moraes et al. found that general-purpose LLMs produced NCL code that violated syntax or requirements despite plausible-looking output [17]. Although their task is code generation, it reinforces Web-specific artifacts and expert validation. ArcAssist-Web transfers that lesson from generated Web code to architectural evolution: a plausible recommendation is insufficient unless its project evidence, Web boundaries, assumptions, and proposed checks remain inspectable.

Gap. GenAI can retrieve architectural knowledge [5], supplement sparse descriptions [10], and still perform poorly on quality attributes and rationale [21]. ArcAssist-Web jointly organizes baseline and Web/domain context, generation, validation evidence, and architect acceptance while leaving models, RAG, agents, and checks open.

4 ARCASSIST-WEB LOGICAL REFERENCE ARCHITECTURE

ArcAssist-Web organizes reusable responsibilities, decision artifacts, constraints, variation points, and information dependencies for GenAI-assisted Web evolution; technologies remain open [1, 13, 18, 23].

Inputs and seven responsibilities. Figure 1 receives an architect, concern, baseline, and Web/domain evidence as information sources. Table 1 defines the seven responsibilities; Architect Review is an external acceptance gate. The colored clusters are narrative groupings, not layers or subsystems.

Used together, framing and context identify the change and relevant Web/domain boundaries; knowledge and orchestration ground LLM reasoning; Validation and QA prepares checks. Only an accepted package updates Knowledge Management with ADRs, evidence, and reusable prompts [5, 11, 12, 14].

Decision Package as a review boundary. A Decision Package combines the concern, evidence references, alternatives, consequences, assumptions, and validation plans. This logical handoff separates generated material from the architect's authority to accept, reject, or refine it [11, 12, 14]. A package is inspectable when its scope, rules, alternatives, and validation actions cite identifiers from the frozen manifest; uncited claims remain assumptions for architect review. The evidence link supports challenge and correction, but is not treated as proof that a recommendation is architecturally correct.

The evaluation uses JSON with a required `webContext` object. Its four Web fields are `interactionFlow`, `boundaries`, `dataVisibility`, and `qualityAttributes`. One recorded package keeps public animal-profile responses limited to public *Animal* data, excludes applicant data, and plans an API-contract check against leakage into public responses.

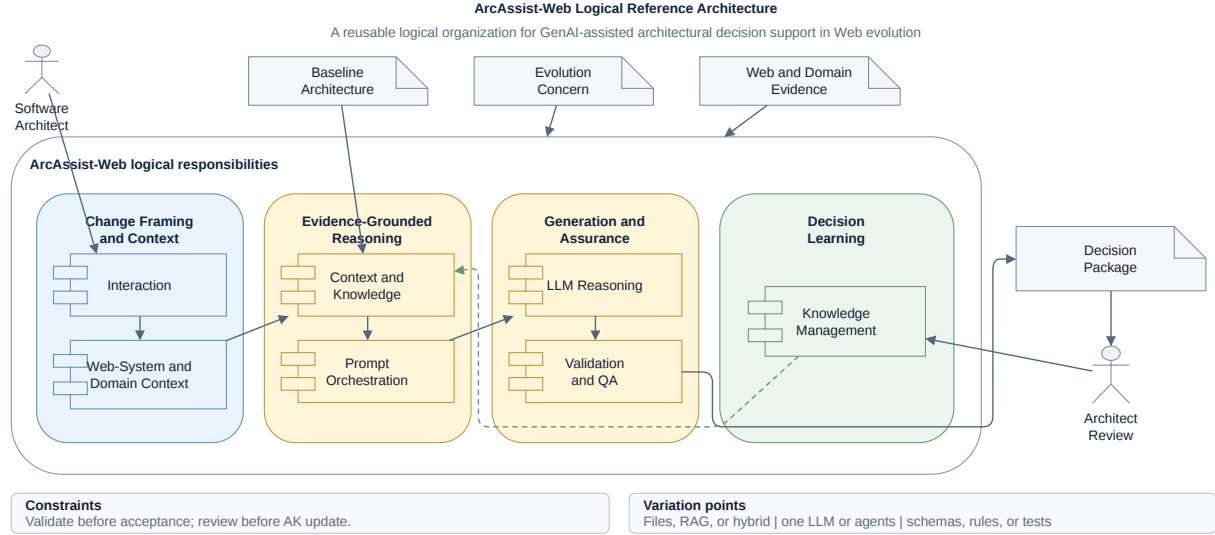
The layout denotes dependencies, not execution order. Validation precedes acceptance and review precedes knowledge updates; files/RAG, one LLM/agents, and schemas/rules/tests remain variation points. In a concrete instantiation, the architect first selects the baseline artifacts and assigns stable evidence identifiers. Prompt or agent workflows may realize orchestration and reasoning, while schemas, rules, and tests realize Validation and QA. ArcAssist-Web does not prescribe these technologies; it prescribes the information exchanged among them and preserves external architect review before any generated proposal becomes retained knowledge.

5 APPLICATION SCENARIO

The application scenario is an animal-rescue Web/PWA that publishes animals available for adoption and gives staff an administrative *Animals* module. Its baseline uses Next.js/PWA, NestJS services, and shared domain, UI, and infrastructure packages. The increment lets a visitor select an animal, submit initial interest, and leave later handling to staff; authentication, donations, screening, and complete adoption remain out of scope. The concrete user goal is deliberately narrow: from an animal's public profile, a visitor records interest and

Table 1: Seven logical ArcAssist-Web responsibilities. Architect Review is an external acceptance gate.

Logical responsibility	Role
Interaction	Frames the Evolution Concern, stakeholders, scope, and quality constraints [11, 12].
Web-System and Domain Context	Builds a Web view from interaction flows, UI/API/data boundaries, data visibility, roles, Web quality attributes, and domain rules [9, 12].
Context and Knowledge	Uses Baseline Architecture and retained knowledge to ground reasoning [5, 21].
Prompt Orchestration	Coordinates grounded inputs for LLM Reasoning [2].
LLM Reasoning	Drafts alternatives, trade-offs, risks, and rationale for a Decision Package [6, 7].
Validation and QA	Checks alternatives, rules, boundaries, and traceability before Architect Review [16, 20].
Knowledge Management	Retains accepted ADRs, evidence, and reusable prompts [5, 12].
External architect review gate (not a logical responsibility). Accepts, rejects, or refines a decision package before knowledge changes [11, 14].	

**Figure 1: ArcAssist-Web logical reference-architecture proposal. Its boxes denote the seven reusable responsibilities; Architect Review is external. Solid arrows denote information dependencies and feedback; the dashed arrow denotes knowledge reuse, not workflow steps.**

contact information for later staff review. The increment ends at registration; it neither decides eligibility nor changes the animal’s adoption status.

Change framing and context. *Interaction* frames that bounded concern. *Web-System and Domain Context* identifies the *profile* → *interest submission* → *staff handling* flow, public/non-public boundaries, and privacy, modifiability, and traceability constraints. **AVAILABLE** is implemented, but permitted transitions are not; a transition policy remains a proposal. This yields an end-to-end Web slice: profile page and form, submission DTO/API boundary, a non-public *AdoptionInterest* record, and a future protected staff view. The visibility rule prevents applicant data from being returned with the public *Animal* representation.

Evidence-grounded reasoning. *Context and Knowledge* uses seven baseline items: the architecture description; first-iteration scope; *Animal* status values; the absent transition encoding; the public detail page with an unimplemented interest link; the Animals-only API surface; and the planned-concepts diagram. The separate four-item Web/domain group includes the incremental backlog, ADR-002 (layer separation), ADR-003 (versioned ADRs, diagrams, and

reusable prompts), and the absence of an applicant-data model or staff-review UI/API; four explicit constraints or open questions complete the 15 frozen C inputs. They constrain scope and layering and identify *Adoptant* and *AdoptionInterest* as planned, not implemented. *Prompt Orchestration* requests alternatives, Web/domain implications, validation, and documentation updates.

Generation and assurance. *LLM Reasoning* compares deferral, a bounded *AdoptionInterest* boundary linked to *Animal*, and extension of the *Animal* boundary. Configured outputs provisionally recommend the bounded boundary, separating the increment from screening, authentication, and public applicant data. *Validation and QA* plans checks for consent, data visibility, duplicate interests, status policy, and layer separation. The pre-review package includes alternatives, consequences, grounding, validation, and proposed knowledge updates. The provisionally recommended alternative – a bounded *AdoptionInterest* boundary linked to *Animal* – does not claim implementation. It remains a proposal for architect review because its rationale points to the planned *AdoptionInterest* concept, distinguishes current facts from the proposed status policy, and preserves unresolved consent,

retention, duplicate-submission, and authorization questions as validation work.

Decision learning. An architect may accept, reject, or refine the package. Only acceptance lets *Knowledge Management* retain its ADR, diagrams, backlog, evidence, and prompt; the captured outputs stop before review. Accordingly, the scenario demonstrates the complete reasoning handoff, not a deployed feature: inputs are frozen, alternatives are generated, checks are proposed, and acceptance remains outside the recorded run.

6 ILLUSTRATIVE TRACEABILITY EVALUATION

We ask whether configured decision material remains linked to a frozen Web baseline and explicit about validation. Three retained output pairs address one adoption-interest concern; an earlier pilot pair was excluded because output language was not controlled. The comparison evaluates the complete evidence-and-contract configuration, not any single mechanism or architectural correctness.

1. Generate comparable decision material. Pairs were captured on June 24, 2026 with recorded model identifier `gpt-5.5`, medium reasoning, English, and no project browsing. The generic condition (G) received the concern plus instructions to propose one change in Markdown, avoid asserting baseline facts, and label baseline claims as assumptions. The configured condition (C) received the same concern, 15 frozen baseline, Web/domain, constraint, and open-question items, and a contract requiring at least two alternatives in JSON.

2. Inspect package structure. We inspected evidence identifiers, alternatives, consequences, and planned validation. The checker verifies C's JSON contract and identifiers, not architectural correctness. Unavailable provider instructions, temperature, seed, and output cap remain reproducibility threats. For each pair, we recorded whether the output cited the frozen manifest, separated alternatives from consequences, exposed Web context and assumptions, and proposed project-linked checks. We did not score recommendation quality because no independent architect assessed correctness or usefulness.

Table 2: Per-run observations; generic (G) and configured (C) values are shown explicitly.

Aspect	Run 1	Run 2	Run 3
Frozen evidence IDs cited	G=0; C=15	G=0; C=15	G=0; C=15
C JSON contract valid	yes	yes	yes
Alternatives	G=1; C=3	G=1; C=2	G=1; C=3
Planned validation items	G=4; C=6	G=4; C=7	G=4; C=6

JSON, alternative counts, and validation counts are instruction-conformance observations, not condition effects, because the prompts imposed those structures. The generic condition received no frozen evidence identifiers, so G=0 is expected and does not represent a failed citation task. By contrast, C=15 means that the configured output cited all 15 supplied identifiers. These counts measure traceability

coverage only, not citation sufficiency, recommendation correctness, or architectural quality. Their value is auditability: a reviewer can trace a statement to the frozen evidence and challenge the inference.

3. Interpret convergence and limits. One G run independently proposed an *AdoptionInterest* entity and a dedicated command endpoint, with deferred staff handling, privacy, duplicate-submission, and consent concerns. This substantively converges on the configured recommendation—a bounded *AdoptionInterest* boundary linked to *Animal*—even though the generic run received no project evidence. The frozen evidence and contract therefore added traceability and explicitness, not evidence of a better architectural conclusion.

The result is descriptive: no validation plan was executed, no architect coded the outputs, and no usefulness, effort, or implementation-quality measure was collected. The artifact preserves inputs, prompts, raw outputs, and checks. A confirmatory study should repeat the configuration across distinct Web/PWA evolution concerns and ask independent architects to rate baseline fidelity, usefulness, missing risks, and acceptance effort; implementation tests should separately evaluate API behavior and data exposure. This design can test whether traceability reduces review effort or improves risk discovery, rather than merely changing the output format.

7 CONCLUSION

ArcAssist-Web organizes seven reusable responsibilities for GenAI-assisted Web architectural evolution. Together, they expose Web/domain context, evidence, validation, and Architect Review while leaving implementation choices open.

The animal-rescue scenario illustrates how a target Web system supplies baseline and evidence. Configured outputs linked Web context and decision material to frozen evidence, but a generic run reached a similar direction; the result supports inspectability and traceability, not superior architectural conclusions. Future work includes independent architect review, additional Web/PWA scenarios, and decision-support tooling. The next validation step is therefore architect assessment across multiple Web evolution concerns, followed by implementation-level checks for accepted decisions.

ETHICS AND GEN AI USE

Human-subject research and personal-data processing do not apply; no platform source code was used. Under researcher supervision, GenAI served as the evaluated decision-support mechanism and assisted repetitive evaluation and English revision. The artifact¹ includes evidence, prompts, paired outputs, metadata, hashes, contract, and a validation script; local paths and credentials are excluded. The recorded scenario is limited to architectural evidence and does not include applicant records, operational telemetry, or completed adoption decisions.

¹<https://github.com/californi/webarch-eval-artifact/>.

REFERENCES

- [1] Samuil A. Angelov, Paul W. P. J. Grefen, and Dirk Greefhorst. 2012. A Framework for Analysis and Design of Software Reference Architectures. *Information and Software Technology* 54, 4 (2012), 417–431. doi:10.1016/j.infsof.2011.11.009
- [2] Marco Becattini, Roberto Verdecchia, and Enrico Vicario. 2025. SALLMA: A Software Architecture for LLM-Based Multi-Agent Systems. In *2025 IEEE/ACM International Workshop on New Trends in Software Architecture (SATrends)*. IEEE, Piscataway, NJ, USA, 5–8. doi:10.1109/SATrends66715.2025.00006
- [3] Alessio Bucaioni, Martin Weyssow, Junda He, Yunbo Lyu, and David Lo. 2025. A Functional Software Reference Architecture for LLM-Integrated Systems. In *2025 IEEE 22nd International Conference on Software Architecture Companion (ICSA-C)*. IEEE, Piscataway, NJ, USA, 1–5. doi:10.1109/ICSA-C65153.2025.00006
- [4] Mike Cohn. 2004. *User Stories Applied: For Agile Software Development*. Addison-Wesley Professional, Boston, MA, USA.
- [5] Rudra Dhar, Karthik Vaidhyathan, and Vasudeva Varma. 2024. Leveraging Generative AI for Architecture Knowledge Management. In *2024 IEEE 21st International Conference on Software Architecture Companion (ICSA-C)*. IEEE, Piscataway, NJ, USA, 163–166. doi:10.1109/ICSA-C63560.2024.00034
- [6] Tobias Eisenreich, Sandro Speth, and Stefan Wagner. 2024. From Requirements to Architecture: An AI-Based Journey to Semi-Automatically Generate Software Architectures. In *2024 IEEE/ACM International Workshop on Designing Software (Designing)*. ACM, New York, NY, USA, 52–55. doi:10.1145/3643660.3643942
- [7] Marios Fokaefs, Hashim Khan, and Borna Ahmadzadeh. 2025. Can AI Build Systems? An Exploratory Study on Generating Software Architecture with LLMs. In *2025 IEEE International Conference on Collaborative Advances in Software and Computing (CASCON)*. IEEE, Piscataway, NJ, USA, 1–10. doi:10.1109/CASCON66301.2025.00069
- [8] David Garlan and Mary Shaw. 1994. *An Introduction to Software Architecture*. Technical Report CMU-CS-94-166. Carnegie Mellon University.
- [9] Athula Ginige and San Murugesan. 2001. Web Engineering: An Introduction. *IEEE MultiMedia* 8, 1 (2001), 14–18. doi:10.1109/93.923949
- [10] Ahmad Hatahet, Christoph Knieke, and Andreas Rausch. 2025. Generating Software Architecture Description from Source Code Using Reverse Engineering and Large Language Model. In *2025 ACM/IEEE 28th International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. IEEE, Piscataway, NJ, USA, 566–575. doi:10.1109/MODELS-C68889.2025.00080
- [11] James Ivers and Ipek Ozkaya. 2025. Will Generative AI Fill the Automation Gap in Software Architecting?. In *2025 IEEE 22nd International Conference on Software Architecture Companion (ICSA-C)*. IEEE, Piscataway, NJ, USA, 41–45. doi:10.1109/ICSA-C65153.2025.00014
- [12] Anton Jansen and Jan Bosch. 2005. Software Architecture as a Set of Architectural Design Decisions. In *Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture (WICSA)*. IEEE, Piscataway, NJ, USA, 109–120. doi:10.1109/WICSA.2005.61
- [13] Philippe B. Kruchten. 1995. Architectural Blueprints: The “4+1” View Model of Software Architecture. *IEEE Software* 12, 6 (1995), 42–50. doi:10.1109/52.469759
- [14] Qinghua Lu, Liming Zhu, Xiwei Xu, Zhenchang Xing, Stefan Harrer, and Jon Whittle. 2024. Towards Responsible Generative AI: A Reference Architecture for Designing Foundation Model Based Agents. In *2024 IEEE 21st International Conference on Software Architecture Companion (ICSA-C)*. IEEE, Piscataway, NJ, USA, 119–126. doi:10.1109/ICSA-C63560.2024.00028
- [15] Felix Mahr, Konstantin Schmidt, Giulia Angeli, Jörg Franke, and Till Sindel. 2024. A Reference Architecture for Deploying Large Language Model Applications in Industrial Environments. In *2024 IEEE 30th International Symposium for Design and Technology in Electronic Packaging (SIITME)*. IEEE, Piscataway, NJ, USA, 19–23. doi:10.1109/SIITME63973.2024.10814877
- [16] Zahra Moezkarimi, Kevin Eriksson, Albin Alm Johansson, Alessio Bucaioni, and Marjan Sirjani. 2025. Harnessing ChatGPT for Model Transformation in Software Architecture: From UML State Diagrams to Rebeca Models for Formal Verification. In *2025 IEEE 22nd International Conference on Software Architecture Companion (ICSA-C)*. IEEE, Piscataway, NJ, USA, 387–396. doi:10.1109/ICSA-C65153.2025.00061
- [17] Daniel de Sousa Moraes, Polyana Bezerra da Costa, Antonio J. G. Bussan, José Matheus Carvalho Boaro, Carlos de Salles Soares Neto, and Sergio Colcher. 2023. On the Challenges of Using Large Language Models for NCL Code Generation. In *Anais Estendidos do Simpósio Brasileiro de Sistemas Multimídia e Web (WebMedia)*. Sociedade Brasileira de Computação, Porto Alegre, RS, Brazil, 151–156. doi:10.5753/webmedia_estendido.2023.236175
- [18] Elisa Yumi Nakagawa, Flavio Oquendo, and Martin Becker. 2012. RAModel: A Reference Model for Reference Architectures. In *2012 Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture*. IEEE, Piscataway, NJ, USA, 297–304. doi:10.1109/WICSA-ECSA.2012.49
- [19] Dewayne E. Perry and Alexander L. Wolf. 1992. Foundations for the Study of Software Architecture. *ACM SIGSOFT Software Engineering Notes* 17, 4 (1992), 40–52. doi:10.1145/141874.141884
- [20] Riccardo Rubel, Amleto Di Salle, and Alessio Bucaioni. 2025. LLM-Based Recommender Systems for Violation Resolutions in Continuous Architectural Conformance. In *2025 IEEE 22nd International Conference on Software Architecture Companion (ICSA-C)*. IEEE, Piscataway, NJ, USA, 404–409. doi:10.1109/ICSA-C65153.2025.00063
- [21] Mohamed Soliman and Jan Keim. 2025. Do Large Language Models Contain Software Architectural Knowledge? An Exploratory Case Study with GPT. In *2025 IEEE 22nd International Conference on Software Architecture (ICSA)*. IEEE, Piscataway, NJ, USA, 13–24. doi:10.1109/ICSA65012.2025.00012
- [22] Alberto Tagliaferro, Simone Corbo, and Bruno Guindani. 2025. Leveraging LLMs to Automate Software Architecture Design from Informal Specifications. In *2025 IEEE 22nd International Conference on Software Architecture Companion (ICSA-C)*. IEEE, Piscataway, NJ, USA, 291–299. doi:10.1109/ICSA-C65153.2025.00049
- [23] Pedro Henrique Dias Valle, Lina Garcés, Tiago Volpato, Silverio Martínez-Fernández, and Elisa Yumi Nakagawa. 2021. Towards Suitable Description of Reference Architectures. *PeerJ Computer Science* 7 (2021), e392. doi:10.7717/peerj-cs.392