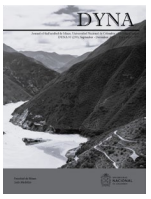
DYNA
http://dyna.medellin.unal.edu.co/



UNIVERSIDAD NACIONAL DE COLOMBIA

ADKT: a support tool for reducing architectural knowledge evaporation in software projects

Santiago Hyun-Dorado[a], Julio Ariel Hurtado-Alegría[a], Enrique Moguel[b] & Jose Garcia-Alonso[b]

[a] *Facultad de Ingeniería Electrónica y Comunicaciones, Universidad del Cauca, Popayán, Colombia. santiagodorado@unicauca.edu.co, ahurtado@unicauca.edu.co*
[b] *Quercus Software Engineering Group, Universidad de Extremadura, Cáceres, España. enrique@unex.es, jgaralo@unex.es*

Received: April 4th, 2025. Received in revised form: September 25th, 2025. Accepted: October 9th, 2025.

Abstract
Defining software architecture is a complex task that demands technical expertise and business knowledge. Design decisions drive architectural structures throughout development and maintenance. However, this knowledge often dissipates over time, a phenomenon known as architectural knowledge evaporation, leading to increased costs and maintenance difficulties. This paper presents ADKT, a support tool designed to mitigate architectural knowledge evaporation by enabling the documentation and traceability of design decisions. ADKT was evaluated through a case study involving software engineers of different expertise levels. The evaluation assessed the tool's ease of use, perceived usefulness, and effectiveness in reducing knowledge evaporation. Results indicate that participants found ADKT easy to use and valuable for preserving architectural knowledge. Nonetheless, challenges remain regarding engineers' commitment to consistently documenting decisions and the need for a designated person to oversee this process. These findings highlight the importance of developing advanced tools and fostering a culture of documentation within development teams.

Keywords: architectural design decisions; software architecture; documentation and traceability; software maintenance; architectural knowledge evaporation.

ADKT: una herramienta para la reducción de la evaporación del conocimiento arquitectónico en proyectos software

Resumen
Definir la arquitectura de software es una tarea compleja que exige experiencia técnica y conocimiento del negocio. Las decisiones de diseño impulsan las estructuras arquitectónicas durante el desarrollo y mantenimiento. Sin embargo, este conocimiento a menudo se disipa con el tiempo, un fenómeno conocido como evaporación del conocimiento arquitectónico, lo que genera mayores costos y dificultades de mantenimiento. Este artículo presenta ADKT, una herramienta de apoyo diseñada para mitigar la evaporación del conocimiento arquitectónico al permitir la documentación y trazabilidad de decisiones de diseño. ADKT fue evaluado mediante un estudio de caso con ingenieros de software de distintos niveles de experiencia. La evaluación analizó la facilidad de uso, utilidad percibida y efectividad de la herramienta para reducir la evaporación del conocimiento. Los resultados indican que los participantes encontraron ADKT fácil de usar y valioso para preservar el conocimiento arquitectónico. No obstante, persisten desafíos en el compromiso de los ingenieros para documentar decisiones de manera constante y la necesidad de una persona encargada de supervisar este proceso. Estos hallazgos destacan la importancia de desarrollar herramientas avanzadas y fomentar una cultura de documentación en los equipos de desarrollo.

Palabras clave: decisiones de diseño arquitectónico; arquitectura de software; documentación y trazabilidad; mantenimiento de software; evaporación del conocimiento arquitectónico.

1 Introduction

Software architecture plays a critical role in the development and evolution of software systems. It encompasses the fundamental structure of a system, defining its components, their relationships, and the guiding principles governing their design and evolution [1]. Architectural design decisions (ADDs) are central to this process, as they

How to cite: Hyun-Dorado, S., Hurtado-Alegría, J.A., Moguel, E., and Garcia-Alonso, J., ADKT: a support tool for reducing architectural knowledge evaporation in Software projects DYNA, (92)239, pp. 56-65, October - December, 2025.

encapsulate the rationale behind the selection of specific patterns, technologies, and organizational structures that shape the software system [2,3].

Despite their importance, ADDs are often poorly documented or entirely omitted from system artifacts [4]. This results in architectural knowledge evaporation, a phenomenon where the rationale and context behind architectural decisions are lost over time [5]. This loss can severely impact software maintainability, increase costs, and hinder future development, as teams struggle to understand the reasons behind existing architectural choices [3,6].

The problem is exacerbated in agile and fast-paced development environments, where teams prioritize rapid delivery over extensive documentation [5,7]. Consequently, architectural decisions are frequently communicated informally through verbal discussions, chat messages, or emails, leaving no traceable record [8]. As the project progresses, team members change, and the context surrounding past decisions dissipates, making it challenging to evolve the architecture without risking architectural drift or degradation [9].

Existing approaches to address architectural knowledge evaporation include both reactive and proactive methods. Reactive approaches focus on mining ADDs from existing artifacts, such as code repositories, emails, and task management systems [10,11]. Proactive methods aim to capture ADDs in real-time through tools like design wikis, decision logs, and chatbots. While these solutions offer valuable contributions, they often lack a unified structure for documenting and tracing ADDs across diverse software artifacts [12].

This work introduces ADKT, a support tool designed to reduce architectural knowledge evaporation by enabling the systematic capture, traceability, and management of ADDs throughout the software development lifecycle. ADKT provides a standardized data model to document ADDs and their relationships with system artifacts, ensuring that architectural knowledge remains accessible and up to date. The tool was validated through a case study with professional software engineers, assessing its ease of use, perceived usefulness, and impact on knowledge preservation.

The rest of the paper is organized as follows: Section 2 reviews related work on architectural knowledge management and traceability. Section 3 presents the ADKT tool and its underlying data model. Section 4 describes the case study methodology and evaluation process. Section 5 discusses the results and their implications. Finally, Section 6 concludes with reflections on the tool's effectiveness and future research directions.

2 Background

The concern about architectural evaporation is not new, and numerous authors have attempted to address it from different angles, some through automated means, others manually. In a previous study [8], we investigated trends to understand the current state of research on approaches for mapping ADDs and how they connect with the various software artifacts involved in the development process. To this end, we conducted a systematic mapping study (SMS) of the literature, following the method by [13], in which the search string, research questions, inclusion and exclusion criteria were defined, and 34 relevant articles were analyzed. The study concludes that most efforts are focused on reactive approaches for developing software tools driven by artificial intelligence (AI) to extract architectural knowledge from sources such as technical documentation, communication tools, source code, and task management systems. This study underscores the importance of preserving this knowledge to mitigate AK evaporation and future architectural erosion, although it also highlights limitations such as subjectivity in determining relevant elements, artifacts, and ADDs, as well as challenges in validating the proposed approaches.

From this first study, some papers of interest in the scope of this paper were extracted. An example of these reactive approaches for AK preservation is presented by [5], where the challenges of Architectural Knowledge Management in the context of global agile software development are discussed. A concept called AK condensation is introduced to reduce the evaporation of AK stored in Unstructured Textual Electronic Media (UTEM) records. This concept was implemented through a prototype known as ArchiKCo, which includes mechanisms for classifying and searching AK. To evaluate the effectiveness of AK condensation, studies were conducted with professional developers and students, showing promising results in retrieving AK from UTEM records. This is an innovative approach because it allows obtaining architectural knowledge from sources familiar to developers; however, it is limited to a restricted set of technologies such as Skype and Windows. Additionally, the collected information lacks a well-defined structure for ADDs, which could lead to finding knowledge that is not architecturally relevant.

In another vein, [14] suggest a more proactive method with a framework to simplify the real-time recording of software design decisions using instant messaging tools and chatbots commonly used by developers. The core proposal is to integrate a chatbot that guides developers in systematically documenting design decisions through natural language processing (NLP). This chatbot collects information in a data template, stores it in a structured database, and later exports it to a wiki using markdown and version control for future reference. However, there is no option to modify or delete archived decisions, the NLP does not effectively detect synonyms or duplicates, the structure of the decisions is linear with no linking between them, and no empirical user study has been conducted to evaluate its perceived usefulness by professionals. It is restricted to a specific set of technologies, and although it has a defined structure for documenting ADDs, it does not facilitate traceability between them or linking with related software artifacts.

Similarly, [15] present KnoCap, an idea consisting of a note-taking tool with a physical button to record design decisions in the form of voice notes during whiteboard meetings. The goal is to facilitate the capture of design decisions during meetings without distracting attendees from their primary function. They have also developed a web application that allows users to access and review recorded voice notes along with automatically generated transcripts. At the time of writing the article, the proposal includes only

the fundamental aspects. This approach could disrupt the flow of meetings due to the need to record important parts, and it is often challenging to reconstruct what was discussed without additional notes, images, or a visible whiteboard, leading to significant loss of knowledge about decisions and architecture. As with other methods, decisions are not recorded in a well-defined format or structure.

Subsequently, the same authors [16] improves KNOCAP, a set of tools for capturing and delivering important design bits (IDBs) from whiteboard meetings. This proposal incorporates lightweight interfaces for designers to highlight crucial audio segments during meetings, intelligent storage of IDBs through voice recognition, indexing and classification of contribution types, and delivery of relevant previous IDBs for future meetings through reactive conversational agents for queries and proactive agents to suggest pertinent IDBs. Among the main contributions are new capture interfaces, advanced IDB storage techniques, and delivery mechanisms through conversational agents. Challenges include the subjective identification of "important" information, avoiding overwhelming designers with excessive suggestions, developing effective dialogue models, and robust language processing capabilities.

Considering the above, there is a lack of a well-defined structure that can be used by different methods as a basis for representing and tracing these ADDs among the various artifacts generated in software development and the decisions themselves, as well as the possibility of collecting this information independently of the relevant source for each author through a common interface. Traceability, in general terms, is the ability to link each component, decision, or change within a system to the requirements, constraints, or prior decisions that support them [17]. This process allows understanding how and why the system was developed in a specific way, providing a solid foundation for impact analysis and change management. In software architecture, traceability encompasses both ADDs and the artifacts produced, such as models, diagrams, documentation, source code, and other elements mentioned that are fundamental for evaluating the coherence between the architecture and quality goals [12]. This is especially relevant when the system must adapt to new demands or changing conditions, as traceability helps identify affected design elements, promoting controlled evolution of the architecture.

Although there are numerous methods for extracting this AK from various sources, there is still no common standard that ensures adequate traceability of ADDs and software artifacts, resulting in the continuous erosion of software architectures, AK evaporation, and additional efforts by development teams to make informed decisions for system evolution. Next, a complementary proposal to existing studies and its subsequent validation through a case study is presented.

3 ADKT: a tool for tracing architectural design decision knowledge

ADKT is presented as a simple initiative to unify and strengthen the efforts of various authors in mitigating architectural knowledge evaporation, a phenomenon previously mentioned that is characterized by the gradual loss of crucial information about ADDs throughout the software lifecycle.

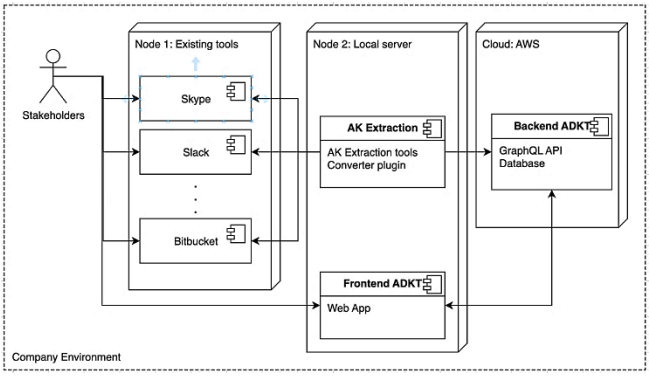


Figure 1. Proposed architecture for the capture from different sources
Source: Own elaboration

In its initial conception, ADKT was designed as a model to efficiently capture and manage relevant information linked to ADDs. However, its scope expanded upon recognizing the importance of the artifacts generated and used during the software development process. This ensures comprehensive traceability of architectural decisions, linking them to system components and key stakeholders involved, such as developers, software architects, and other interested parties.

The fundamental pillars on which ADKT is built are:

- Interdisciplinary Collaboration: Promotes cooperation between authors and teams, fostering a shared vision of the problem.
- Consensus on Traceability Structure: Defines a common and standardized structure for documenting, relating, and tracing ADDs coherently.
- Extensibility through Plugins: Facilitates the integration of additional plugins to extend the tool's functionality, adapting it to various business contexts and specific needs.

One of the main advantages of ADKT lies in its simplicity, making it accessible to both technical and non-technical teams. Additionally, its ease of integration allows for implementation in any business environment, regardless of size or complexity. This makes it a flexible and practical solution, capable of addressing the challenges associated with managing and preserving architectural knowledge in the software industry.

Below, in the following sections, the ADKT proposal is detailed in descending order of abstraction level: the architecture and its components, implementation aspects, and finally, the defined data model. In Fig. 1, the elements that make up the architecture and their interrelationships can be observed, followed by a detailed description of each.

3.1 Node 1 - existing tools

It represents the execution environment where all the programs commonly used by the stakeholders (analysts, designers, developers, architects, sponsors, managers, or anyone else with an interest in the development of the

project) run, including communication tools such as Slack, WhatsApp, Email, Skype or others; task management tools such as Jira, Trello, Excel, etc; the source code itself hosted on GitHub, GitLab, Bitbucket and others version control systems, technical documentation in Figma, Prototypes, Word, Confluence, Docs, among others.

3.2 Node 2 – Local server

This node represents the execution environment within development organizations where programs developed for extracting and documenting ADDs are executed.

3.2.1 AK extraction

Represents the development of various systems for extracting ADDs from existing tools. As mentioned in the background section, task management tools like Jira, communication tools like Skype, Slack, source code, code repositories like GitHub, Bitbucket, technical documentation, UML, requirements, etc., are frequently used as data sources for extracting ADDs using artificial intelligence techniques

3.2.2 Converter plugin

Additionally, each company can develop an additional subcomponent integrated with the AK extraction systems, responsible for formatting the data to the model proposed in this work to load ADD information into the ADKT Backend, aiming to have a record of these decisions in a common database and allow them to be queried in a single system by different stakeholders.

3.2.3 Converter plugin

Its primary function is the manual registration of ADDs through a data form with instructions and examples to facilitate their documentation. It can also be used to query ADDs captured automatically by AK Extraction tools and manually through the form. It can be installed using Docker commands to facilitate its deployment, with prior installation of the ADKT Backend and configuration of some environment variables. For more information, refer to the official documentation [18].

3.3 AWS cloud

A fully managed Serverless application by AWS Cloud. For its installation, the source code must be requested, and a meeting arranged to install the program directly in the interested company's cloud. This can be done through a contact form available in the official documentation [18]. The application is a GraphQL API in AppSync protected with an API KEY.

To make requests, only the API KEY is required, and the HTTP call documentation available on the official site must be followed [18]. In Fig. 2, the elements of the Backend (AppSync, Lambda, DynamoDB) are detailed, as well as the connection with the Frontend (NextJS, Cognito) and the technologies used to the deployment (Netlify).

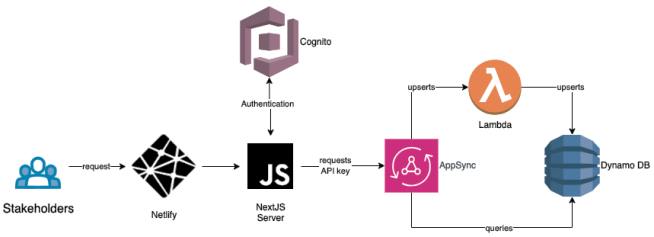


Figure 2. ADKT – Architecture Design
Source: Own elaboration

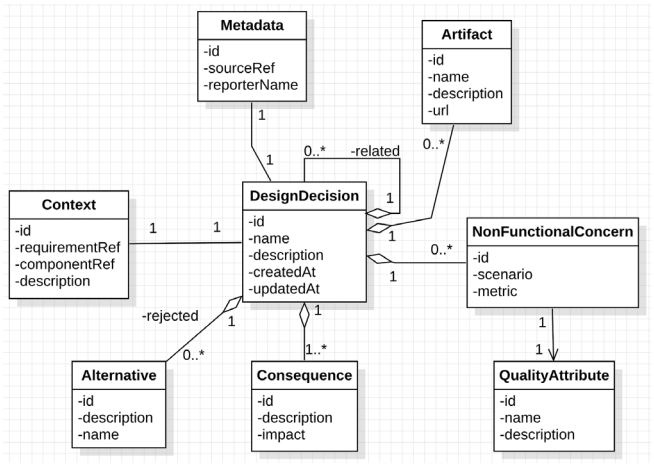


Figure 3. ADKT – Architecture Design
Source: Own elaboration

3.3.1 ADKT model

Data model that represents the knowledge about ADDs, how they relate to each other, and the link with the artifacts generated in the software development process. Below, each of the elements that make up the model is described.

In Fig. 3, you can graphically detail the attributes of each element and the multiplicity of the relationships, which indicates the mandatory nature of each component. This model is an extended version of the design decision metamodel proposed by [14]. Some attributes of the existing entities were modified to make it simpler, and some relationships were added, such as artifacts linked to a specific decision, and a self-relationship that represents the traceability that may exist between certain decisions that depend on previous ones.

- Design Decision: The central element of the model, specifies a name for identification, a detailed description of the selected decision, as well as a creation date, update date, and a link to other related design decisions.
- Metadata: This element allows knowing who registered the decision and the source of the registration. Depending on the plugin used to register the action, a different source reference will be linked.
- Context: As part of each decision, the context is a fundamental element for its understanding as it provides relevant information about the conditions under which a decision is made. Additionally, it links a reference to the requirement motivating the decision. This reference can

be a unique number, a user story code, or any other identifier linking the requirement. Finally, it also links a reference to the component affected by the decision, which can be the name of a service, subsystem, specific class, package, or any other unit or module.

- Alternatives: In the analysis for deciding, the different alternatives that will be compared when selecting must always be considered. The alternatives consist of a name for identification and a description of the reasons why they were discarded during the decision-making process.
- Consequences: Every decision entails at least one consequence, so it is vital to fully identify these effects when making the decision, as designers will have to address them. These consequences have both negative and positive impacts, so a description of that impact is required.
- Artifacts: To maintain the traceability of design decisions, they must not only be captured but also linked to other related documents, resources, or artifacts. These consist of a name for identification, a description indicating the type of artifact, and a URL to locate the resource.
- Non-Functional Concerns: Generally, decisions related to architectural aspects are linked to non-functional requirements (NFR). These consist of scenarios that mention: the stimulus and its source, a metric allowing its measurement, and the quality attribute in question. These quality attributes can be managed independently depending on the quality model followed by each organization.
- Attributes: These are directly linked to NFRs and, unlike them, do not describe metrics or scenarios. Essentially, they consist of a name and description, and for reference, the web platform advises following the names of the product quality model defined by ISO/IEC 25010 [19].

4 Evaluation of ADKT through a case study

According to the systematic mapping study presented by [8], it is important to note that a large percentage of the proposals presented up to the date of the study do not include a validation method for the proposed approaches, methods, or tools. This may be due to the complexity of the proposals and the time available for conducting empirical validation. However, ADKT presents a simple complementary option that can be integrated with existing works, and its validation can also be straightforward through the proposed tools, the model, and a short case study that allows for quantitative and qualitative conclusions. Below, the design of the case study is described.

4.1 Case study design

For the development of this case study, the guidelines for conducting and presenting case studies in software engineering, proposed by [20], were followed. The work focuses on a holistic explanatory case study, as the goal is to explain the impact of the new ADKT proposal in reducing AK loss within the context of a software development project. Below, the unit of analysis, the study objective, the research questions to be addressed, the protocol and instruments for data collection, and finally the analysis process to clarify the results are described in detail.

4.1.1 Unit of Analysis

The Quercus Software Engineering Group research group at the University of Extremadura in Spain [21]. This group is responsible for managing research and development projects in quantum computing. Some of its members are dedicated to developing software components to provide traditional and quantum computing services, while others focus on purely administrative and management activities. Currently, they do not follow any specific methodology or framework such as SCRUM, XP, or RUP. However, they do hold weekly planning meetings for task assignment, goal definition, and deliverables, as is common in any software development team. They are currently in the design and construction phase of services, making it an ideal stage for the inclusion and evaluation of the ADKT tool.

4.1.2 Case study objective

The objective of this case study is to explore the impact of ADKT on reducing AK evaporation during the management of ADDs in the context of software component development. On one hand, it quantitatively evaluates whether there is a relationship between the use of the ADKT model and the reduction of architectural evaporation. On the other hand, it qualitatively analyzes the Perceived Usefulness and Perceived Ease of Use of the proposed model along with the provided web tool.

4.1.3 Research questions

The following research questions are aimed at achieving the research objective:

- RQ1: How do engineers document ADDs in their normal workflow before being introduced to ADKT?
- RQ2: How do engineers perceive the ease of use of the ADKT tool?
- RQ3: Do engineers find the ADKT tool useful in their normal development process?
- RQ4: What are the challenges engineers face when introducing new documentation tools into their workflow?
- RQ5: What are the limitations to consider when reducing architectural evaporation?
- RQ6: How does ADKT impact the reduction of AK evaporation?

The main objective of RQ1 is to characterize how engineers manage the documentation of ADDs, including the tools, models, ceremonies, or any other aspect relevant to preserving ADDs. RQ2 and RQ3 are directly related to determining the ease of use and perceived usefulness of the tool by the case study participants. The purpose of RQ4 is to determine the willingness and adoption protocol of new technologies by the development team. Finally, RQ5 allows us to delve deeper into how these limitations can guide new research to overcome them and provide more information to

move closer to the goal of reducing architectural knowledge evaporation. RQ6 provides clarity on the relationship between ADKT and AK evaporation.

4.1.4 Materials

The following documents and tools [22] were developed to support the research objective and were used during the execution of the case study:

- Architecture Questionnaire: A Google Form Questionnaire containing relevant questions for the study, divided into two sections: The first section consists of 10 questions about personal and demographic data to characterize the participants. The second section includes 16 questions related to architecture, aiming to provide engineers with a starting point to make the implicit information they have about ADDs explicit and available for documentation in their own artifacts or in the ADKT tool if they prefer.
- Logbook: A Google Sheets Document with three tabs to record ADKT usage times, incidents, and any questions that arise during the process. This serves as a record of events throughout the case study.
- Semi-structured Interview: Semi-structured interviews are conducted with each participant to extract qualitative data about the model, the tool, and the process carried out during the study. The initial interview questions are listed below. However, additional questions or comments that arise during the interview will also be considered in the study discussion.
- How is the documentation of ADDs currently carried out?
- What difficulties have you encountered when using ADKT?
- What benefits do you find in using ADKT?
- How many and which decisions have been recorded using ADKT?
- What benefits do you find in documenting ADDs?
- Would you use ADKT in the future, and why?
- TAM Survey: A Technology Acceptance Model (TAM) [23, 24], is used to measure the perceived usefulness and ease of use of ADKT by engineers. This model is implemented in a Google Form with four sections: the first section includes personal data, the second section consists of 10 questions assessing perceived usefulness, the third section contains 8 questions evaluating perceived ease of use, and the final section includes additional comments from participants.

4.1.5 Protocol

The first step of the protocol involves notifying all case study participants via email about the existence of ADKT, along with a link to the web tool. Following this, a semi-in-person meeting is scheduled with the participants to explain each of the materials in detail. During the meeting, training on the tool is conducted through direct interaction and demonstrations with examples.

Subsequently, participants are asked to complete the Architecture Questionnaire, which is crucial for clarifying ideas about the current state of the architecture and establishing a foundation for using the tool. Next, the logbook is introduced, where participants can enter any questions that arise during the study, record usage time, and report any incidents with the tool.

A week after the installation and introduction of ADKT, semi-structured interviews are conducted individually with each participant. Once the ADKT trial period concludes, the study is finalized with the TAM survey to determine participants' perceived usefulness and ease of use of the tool.

4.2 Case study execution

4.2.1 Architecture questionnaire

As mentioned earlier, the first part of the questionnaire gathers demographic information about the participants to save time and minimize interruptions to the team's normal workflow with additional questionnaires. The second part consists of architecture-related questions, which can serve as a basis for documenting previously undocumented decisions. The results are described below.

The group consists entirely of six men, researchers, and software developers between the ages of 23 and 38. Their academic levels range from undergraduate to doctoral degrees, with industry experience spanning from six months to 15 years. They work with conventional technologies and programming languages such as Python, Java, NodeJS, AWS, Flask, Kafka, and Azure, as well as libraries like Qiskit and Braket for quantum circuit development. Their domains of expertise include education and research, healthcare, innovation and product development, and communications. Some members are also faculty at the same institution, teaching courses related to computer science and software development.

Below are the responses to some of the most relevant architecture-related questions regarding the documentation and communication of Architectural Design Decisions (ADDs):

- How are architectural changes managed? Architectural change management includes keeping old local versions for recovery, team discussions, and collaborative decision-making. Changes are discussed in weekly meetings iteratively; however, there is no formal process in place.
- Is there a formal process for making architectural design decisions? (If so, please describe it?): No, typically, a team member proposes an idea, and the developer implements it as they see fit, asking colleagues questions to ensure the implementation aligns with previously established guidelines.
- How is the architecture communicated to stakeholders? Verbally in weekly meetings and sometimes through messaging applications like Slack and email.
- What architectural artifacts are used in the project (e.g., diagrams, models, documents, websites, etc.)? Diagrams, some digital and others on paper, as well as code repositories.
- What challenges does the team face regarding the current architecture? There is barely any documentation on changes and architectural components. If I left my job right now, no

one would have access to the code or past versions, nor would they know how it works. The only preserved artifacts would be paper diagrams outlining the architecture at a high level. Process improvements, inherent challenges in quantum computing, and optimization of waiting times and task execution are needed.

- What are the team's priorities for improving the architecture in the future? We function relatively well, but we lack traceability and documentation, which is an area for significant improvement. The main priority is enhancing processes, especially communication workflows.

4.2.2 Logbook

No incidents or additional questions were recorded. However, the usage times shown in Table 1 were logged, documenting the registration of three ADDs by a single participant.

Although the decisions were recorded during the case study, it is impossible to know with certainty whether they had a positive impact at the time of recording, as the usefulness of the tool is usually seen over time. However, according to the comments received in the TAM Survey, these decisions reflected the reasoning behind certain functional and quality characteristics, which can provide greater context when refactoring the architecture proposed for the project in subsequent design and planning meetings of the Quercus research group, even for participants who were not part of the initial design.

4.2.3 Semi-structured interview

The comments received in the semi-structured interview are discussed in the Results section.

4.2.4 TAM survey

Table 2 presents the average scores obtained after administering the TAM survey to study participants. The complete survey results are available in the repository for this article [22]. For a discussion of the results, please refer to the next section.

Table 1.
Decisions and times registered.

Participant	Start	End	Date	Decision
Participant2	12:30PM	12:42PM	24/05/2024	Functionality
Participant2	8:25AM	8:37AM	27/05/2024	Accessibility
Participant2	12:40PM	12:55PM	06/05/2024	Modularity

Source: Own elaboration

Table 2
TAM survey results.

Participant	Perceived Usefulness	Ease of Use
Participant1	4.40	4.00
Participant2	4.20	3.88
Participant3	4.20	3.00
Participant4	4.40	3.63
Participant5	4.70	3.88
Participant6	3.00	3.00
Total	4.15	3.56

Source: Own elaboration

5 Analysis of results

In this section, the results obtained are analyzed both quantitatively and qualitatively, addressing each research question posed in the experiment design. Finally, a discussion on these results and how they contribute to the existing alternatives for traceability of ADDs and artifacts in software development is presented.

5.1 RQ1: How do engineers document ADDs in their regular workflow before being introduced to ADKT?

Due to the informality of the development methodology, there is no specific method, process, ceremony, or technological tool dedicated to documenting ADDs. However, some ADDs are briefly noted in a participant's notebook during planning ceremonies.

When ADDs arise outside meetings, they are not documented and instead are reflected in artifacts such as source code, Slack messages, or emails. Additionally, in many cases, this information is shared verbally without leaving any structured record.

5.2 RQ2: How do engineers find the usability of the ADKT tool?

Based on semi-structured interviews, all participants generally considered the tool easy to use due to its simple interface, which does not require advanced technical knowledge. Below are paraphrased comments from participants: The information indicators for each attribute help understand what data should be associated with each model element. The tool is easy to manage, and the form is intuitive; three design decisions were registered without issues or additional questions. In contrast, the TAM results in Table 2 reveal that most participants perceived the tool as easy to use. However, only one participant performed the registrations, while the others merely observed. Additionally, two of the lowest results were due to participants not using the tool because of lack of time, priority, or because they forgot. This is corroborated in the additional survey comments [22].

5.3 RQ3: Do engineers find the ADKT tool useful in their regular development process?

Overall, all interviewees indicated in the semi-structured interviews that they considered the tool beneficial. Specific comments included: It helps to have a broader view of ongoing activities, improves documentation of decisions that were previously not recorded, is useful to have this information available and centralized when writing a research paper to save time, could be used to document decisions unrelated to architecture, helps preserve knowledge when people leave the project, allows keeping a history of decisions, it is beneficial that one person's decisions are accessible to the entire group, and it is easier to have the information in the tool than to ask someone directly for a summary. These positive aspects align with the TAM results presented in Table 2. However, it is important to note that

only one participant registered design decisions, while the others primarily acted as observers. Additionally, one participant neither used the tool nor perceived its utility.

5.4 RQ4: What challenges do engineers face when introducing new documentation tools into their workflow?

A fast-paced work rhythm, constantly changing goals, lack of a clear direction when updating research project priorities, and team dynamics have hindered the adoption of tools, methods, or work models. Previously, attempts were made to implement the SCRUM methodology; however, ceremonies could not be held, nor could tasks be formally tracked. Additionally, the nature of the research group prevents fully focusing efforts on a single project. Depending on current research needs, some development projects may be temporarily paused while other objectives, such as publishing papers, are achieved. Lightweight tools like Slack have been useful in keeping the team informed outside planning meetings, where information is shared verbally to avoid using other tools and interrupting the meeting flow.

5.5 RQ5: What limitations should be considered when reducing architectural drift?

Time constraints and work dynamics make it difficult to document design decisions. Constant changes in objectives and priorities affect the dedication to this task. Despite recognizing its importance, it is often forgotten or unclear how to perform it. Many details are omitted during the recording process because they are considered self-evident. Although automated tools help, human intervention is required to ensure the relevance of the information.

5.6 RQ6: How does ADKT impact the reduction of AK drift?

Semi-structured interviews, along with a follow-up meeting after the case study, revealed additional ADDs not recorded in ADKT and confirmed that no decisions with architectural impact were made during the study. This indicates that at least three ADDs were documented during the project, which might not have been recorded otherwise, as mentioned earlier and confirmed by participants' comments. However, the amount of information on ADDs may be insufficient for future decisions, as the complete ADKT model was not used. In the Record Calculations [22], it can be observed that for the first decision, 60% of the data was documented, including the name, description, context, a consequence, and an alternative. For the second decision, in contrast to the first, an artifact was included instead of alternatives, reaching 67%. Finally, for the third decision, no artifacts or alternatives were recorded, resulting in 47% of the information being documented.

Beyond the quantitative percentages of documentation, the impact of recording these three decisions (Functionality, Accessibility, and Modularity) is relevant for the project. For instance, the explicit trace of the Functionality decision allows future developers to understand why certain features were prioritized, reducing the risk of redundant work when new requirements appear. Similarly, the Accessibility decision highlights quality concerns that might otherwise be overlooked in subsequent iterations, helping to align technical improvements with user-centered goals. Finally, documenting Modularity provides a rationale that can guide future refactoring efforts, making it easier for new members of the team to understand the decomposition criteria applied. Although these benefits will be more evident in the medium and long term, the case study shows that even a small number of recorded decisions can provide valuable context for maintaining architectural coherence, supporting onboarding of new developers, and facilitating informed changes during project evolution.

5.7 Discussion

The findings indicate that ADKT effectively helps reduce AK evaporation due to its simplicity and usefulness, as recognized by the stakeholders. Quantitatively, the ADDs collected during the case study were recorded using the tool and prove useful for future stakeholders involved in the project. Qualitatively, participants stated that ADKT is easy to use and expressed their intention to continue using it to document future ADDs. Compared to previously mentioned approaches, it is worth highlighting that this proposal establishes a clear structure for documenting ADDs, including not only the information about the ADD itself but also the traceability between them and the artifacts associated with the decision-making process.

In addition to offering manual data entry, this tool provides a comprehensive architecture designed to integrate with other existing approaches via a well-documented API that runs in the AWS cloud with serverless services that enable automatic scalability of the solution. This allows for the integration of widely used software development tools such as Jira or Confluence, where calls to external APIs can be automated with relevant information from tickets or shared project documentation. It also allows for communication tools such as Slack, where API calls can be made from the definition of specific events in the tool's configuration. The complete installation of this API can be discussed in a scheduled meeting if other developers wish to implement it. This could benefit researchers or developers who gather this knowledge from various artifacts but have yet to develop an application that integrates and visualizes it for all types of stakeholders. The reduction of AK demonstrates that ADKT positively impacts architectural knowledge retention among development teams.

To optimize ADKT, participants provided recommendations, such as adding a notification system to remind users to record ADDs, requesting feedback when completing a record, including a comprehensive example in the field descriptions, and customizing the model according to the specific needs of companies. These suggestions could be implemented in future updated versions of the tool. Meanwhile, delegating the task of reminding team members to record ADDs to one team member or, in the case of a scientific validation, to the experience evaluator, could be considered.

As a recommendation for future research, it is suggested to integrate the proposed architecture into various organizations developing products that extract this information from different artifacts involved in software development. It is necessary to assess whether consolidating this information in a common and easily accessible place significantly impacts time savings and reducing AK loss.

Beyond the specific case study, the applicability of ADKT can extend to organizations that already use tools such as Jira, Confluence, or Slack. In these contexts, ADKT can act as a complementary layer that centralizes architectural decisions automatically extracted from tickets, project documentation, or communication logs, ensuring that rationale is not lost across distributed platforms. Moreover, its serverless design in AWS and the use of a GraphQL API make the solution inherently scalable, enabling adoption both in small research teams and in large, distributed enterprises. This opens the possibility for ADKT to serve not only as a support tool for academic environments but also as a practical solution for industrial software projects where traceability and knowledge retention are critical.

5.8 Limitations of the study

Regarding ADKT: as it is a manual record, it requires the commitment of participants. Likewise, the level of detail required in the forms may discourage frequent use of the tool, leading to temporary and discontinuous use. On the other hand, the system still lacks automatic mechanisms, such as reminders or notifications, that encourage the continuous documentation of ADDs. However, this is mitigated through the development of the API, where this registration can be automated and integrated with existing tools. Additionally, future work aims to integrate periodic notification features to encourage its continuous use and flexible forms for easy completion.

Regarding the design of the case study: it was developed with a single development group (Quercus SEG), which limits the generalization of the results to other organizational contexts. In addition, the duration of the study was relatively short, which prevented the evaluation of whether the observed benefits are maintained in the medium and long term. Finally, longer projects and distributed teams were not considered, so the findings are limited to a controlled and small-scale scenario. To mitigate these limitations, new case studies with other types of organizations and a larger study population are proposed.

Finally, regarding the methodology: although quantitative and qualitative instruments (semi-formal surveys and the TAM model) were used, the evaluation scale was limited, with a small number of participants. Likewise, the study does not allow for verification of the future usefulness of the documented ADDs, as there is no longitudinal analysis that considers their application in subsequent architectural decisions.

To provide greater transparency, the limitations can be grouped into three categories. (1) Tool-related limitations: ADKT still depends on manual registration, does not include reminders or notifications, and its current forms may discourage continuous use. (2) Case study limitations: the evaluation was conducted with a single research group, with a small number of participants and a short observation period, which restricts the generalization of the findings. (3) Methodological limitations: the instruments applied (semi-structured interviews and TAM survey) provided useful but limited insights, and the absence of a longitudinal analysis prevents confirming the long-term usefulness of the recorded decisions. This categorization clarifies the scope of the study and facilitates future replications or extensions.

6 Conclusions and Future Work

This study examines the issue of architectural knowledge loss, highlighting its significant consequences such as excessive maintenance and additional costs due to the misunderstanding of previous architectural decisions, which may be altered by new decisions that do not consider the original context. It also describes how the scientific community addresses this issue from various perspectives and how different software development artifacts are used as sources of information, with AI tools to extract this knowledge. It is emphasized that these various approaches are innovative and contribute significantly to mitigating the problem; however, there remain certain practical gaps that this research aims to address. These include the lack of a defined structure for ADDs that allows both their registration and traceability, and the absence of a common architecture to integrate existing developments within the scientific community.

In response, ADKT is presented: a tool for the traceability of ADDs and software artifacts. This tool consists of a service-based architecture that facilitates the registration of ADDs through a well-defined API and a user interface, following a data model that connects the ADDs and the involved software artifacts. This approach differs from others because: on the one hand, it is manual, for the use of all the roles involved in the development of a software project, and automatic because it seeks to integrate with existing approaches that use other automated methods for knowledge extraction. Additionally, it includes a consistent data model that standardizes and allows the traceability of the ADDs and artifacts.

To validate this tool, a case study is designed and carried out with the development team of the Quercus SEG group at the University of Extremadura. The study aims to evaluate the perceived usefulness and ease of use of the tool, as well as its relationship to the reduction of architectural knowledge evaporation. The tool is previously installed in the group's development environment, participants are trained, and the study begins. Semi-formal surveys and a TAM are used to quantitatively assess the ease of use and usefulness of the tool. To measure its impact on reducing evaporated architectural knowledge, the ADDs are registered in the tool. This study is a first step in the validation of this proposal. Subsequently, further validations will be carried out with more participants and with longer-term projects.

The results indicate that the tool is perceived as useful and easy to use by participants, and it helps document at least three design decisions that would not have been recorded without ADKT. This suggests that the tool contributes to the reduction of architectural knowledge evaporation, although its effectiveness depends on frequent and correct use by users, which emphasizes the importance of active participation and the need for the tool to provide mechanisms that promote its continuous use.

In conclusion, the development of technologies to document and recover this knowledge is crucial and represents a current concern to avoid excessive maintenance and additional costs in software projects. However, without the habit of recording even the minimal amount of this knowledge, these tools cannot reach their potential as reliable sources for future decisions. Therefore, it is recommended to develop both the necessary technology, and the habit of documenting ADDs, aligned with the tools developed for this purpose. Future work includes the integration of the architecture with existing tools, improvement of features suggested by the study participants, and long-term validation to assess the usefulness of documented ADDs in future decisions.

References

[1] Bass, L., Clements, P., Kazman, R., Software Architecture in Practice. Second. Addison Wesley; 2003.

[2] Harrison, N., and Avgeriou, P., How do architecture patterns and tactics interact? a model and annotation. Journal of Systems and Software, 83(10), pp. 1735-1758, 2010. DOI: https://doi.org/10.1016/j.jss.2010.04.067

[3] Jansen, A., and Bosch, J., Software architecture as a set of architectural design decisions. 5th Working IEEE/IFIP Conference on Software Architecture (WICSA'05), 2005, pp. 109-120. DOI: https://doi.org/10.1109/WICSA.2005.61

[4] van-der-Ven, J.S., Jansen, A.G.J, and Bosch, J., Design decisions: the bridge between rationale and architecture. In: Dutoit, A.H., McCall, R., Mistrík, I., and Paech, B., eds. Rationale Management in Software Engineering. Springer Berlin Heidelberg, 2006, pp. 329-348. DOI: https://doi.org/10.1007/978-3-540-30998-7_16

[5] Borrego, G., Morán, A., Palacio, R., Vizcaíno, A., and García, F., Towards a reduction in architectural knowledge vaporization during agile global software development. Inf. Softw Technol, 112(6), pp. 68-82, 2019. DOI: https://doi.org/10.1016/j.infsof.2019.04.008

[6] Capilla, R., Jansen, A., Tang, A., Avgeriou, P., and Babar, M., 10 years of software architecture knowledge management: practice and future. Journal of Systems and Software. (116), pp. 191-205, 2016. DOI: https://doi.org/10.1016/j.jss.2015.08.054

[7] Hadar, I., Sherman, S., Hadar, E., and Harrison, J., Less is more: architecture documentation for agile development, in: 6th International Workshop on Cooperative and Human Aspects of Software Engineering, CHASE, Proceedings, 2013, pp. 121-124.

[8] Hyun, S., and Hurtado, J., Traceability of architectural design decisions and software artifacts: a Systematic Mapping Study. Foundations of Computing and Decision Sciences, 48(4), pp. 401-423, 2023. DOI: https://doi.org/10.2478/fcds-2023-0018

[9] Tofan, D., Understanding and supporting software architectural decisions: for reducing architectural knowledge vaporization. University of Groningen, 2015.

[10] Dutoit, A.H, McCall, R., Mistrík, I., and Paech, B., Rationale management in software engineering: concepts and techniques. In: Dutoit, A.H., McCall, R., Mistrík, I., Paech, B., eds., Rationale Management in Software Engineering. Springer Berlin Heidelberg, 2006, pp. 1-48. DOI: https://doi.org/10.1007/978-3-540-30998-7_1

[11] Bass, L., Clements, P., Nord, R., and Stafford, J., Capturing and using rationale for a software architecture. In: Dutoit, A.H., McCall, R., Mistrík, I., Paech, B., eds., Rationale Management in Software Engineering. Springer Berlin Heidelberg, 2006, pp. 255-272. DOI: https://doi.org/10.1007/978-3-540-30998-7_12

[12] Kamalabalan, et al., Tool support for traceability of software artefacts. In: Moratuwa Engineering Research Conference (MERCon), 2015, pp. 318-323. DOI: https://doi.org/10.1109/MERCon.2015.7112366

[13] Petersen, K., Vakkalanka, S., and Kuzniarz, L., Guidelines for conducting systematic mapping studies in software engineering: An update. Inf Softw Technol, 64, pp. 1-18, 2015. DOI: https://doi.org/10.1016/j.infsof.2015.03.007

[14] Gilson, F., Annand, S., and Steel, J., Recording software design decisions on the fly. CEUR Workshop Proc., (2799), 2020, pp. 53-66.

[15] Meza-Soria, A., and Van-Der-Hoek, A., Collecting design knowledge through voice notes. Proceedings IEEE/ACM 12th International Workshop on Cooperative and Human Aspects of Software Engineering, CHASE, 2019, pp. 33-36. DOI: https://doi.org/10.1109/CHASE.2019.00015

[16] Soria, A., KNOCAP: capturing and delivering important design bits in Whiteboard design meetings. Proceedings ACM/IEEE 42nd International Conference on Software Engineering: Companion, ICSE-Companion, 2020, pp. 194-197. DOI: https://doi.org/10.1145/3377812.3381397

[17] Souali, K., Rahmaoui, O., and Ouzzif, M., An overview of traceability: definitions and techniques. Colloquium in Information Science and Technology, CIST., 2016, pp. 789-793. DOI: https://doi.org/10.1109/CIST.2016.7804995

[18] Hyun, S., ADKT Documentation web site. Preprint posted [online], 2025. Available at: https://zahydo.github.io/adkt-docs

[19] ISO/IEC. ISO/IEC 25010, Systems and Software Engineering-Systems and Software quality requirements and evaluation (SQuaRE)-System and Software Quality Models. ISO/IEC; 2010.

[20] Runeson, P., and Höst, M., Guidelines for conducting and reporting case study research in software engineering. Empir Softw Eng., 14, pp. 131-164, 2009. DOI: https://doi.org/10.1007/s10664-008-9102-8

[21] Quercus Software Engineering Group research group. Preprint posted [online], 2025. Available at: https://quercusseg.unex.es/

[22] Hyun-Dorado S., Materiales para la evaluación de ADKT a través de un estudio de caso. Zenodo. Preprint posted, Art. 9845, 2024. DOI: https://doi.org/10.5281/zenodo.13209845

[23] Davis, F., Bagozzi R., and Warshaw P., User acceptance of computer technology: a comparison of two theoretical models. Management Science, 35(8), pp. 982-1003, 1989.

[24] Wang, C., Ahmad SF, Bani Ahmad Ayassrah AYA, et al. An empirical evaluation of technology acceptance model for Artificial Intelligence in E-commerce. Heliyon, 9(8), e.18349, 2023. DOI: https://doi.org/10.1016/j.heliyon.2023.e18349

S. Hyun-Dorado, is a BSc. Eng. in Systems Engineer from the University of Cauca, Colombia with over 7 years of experience in software development. He is currently finishing his MSc in Computer Science in the University of Cauca. His research interests include Software Engineering, Web Engineering, Software Architecture.
ORCID: 0000-0001-7996-7641.

J.A. Hurtado, is a titular professor at the University of Cauca, Colombia. He completed his PhD in Computer Science from the University of Chile, 2012. His research interests include Software Reuse (Model Driven Engineering, Software Product Lines), Software Architecture, Software Process Modeling and Computational Thinking. Currently he is an active member of the IDIS Research Group.
ORCID: 0000-0002-2508-0962

J.E. Mogel, is an associate professor from the University of Extremadura, Badajoz, Spain. MSc. in Computer Science in 2011, from the University Carlos III, Spain), and PhD in Computer Science in 2018, from the University of Extremadura Spain. His research interests include Web Engineering, Smart Systems, and Quantum Computing.
ORCID: 0000-0002-4096-1282

J. Garcia-Alonso, is currently an associate professor with the Department of Informatics and Telematics System Engineering, University of Extremadura, Badajoz, Spain, and a co-founder of the startups Gloin and Viable. His research interests include quantum software engineering, eHealthCare, eldercare, mobile computing, context-awareness, and pervasive systems. PhD with European Mention in 2014, from the University of Extremadura, Spain.
ORCID: 0000-0002-6819-0299