FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

AI-Based Tool for Lifecycle Management and Preservation of Architectural Questions

Diogo Azevedo Lemos

Mestrado em Engenharia Informática

Supervisor: Ademar Aguiar (FEUP)

Co-supervisor: Neil Harrison (UVU)

July 15, 2025

AI-Based Tool for Lifecycle Management and Preservation of Architectural Questions

Diogo Azevedo Lemos

Mestrado em Engenharia Informática

Approved by:

President: Professor Filipe Figueiredo Correia
Faculdade de Engenharia da Universidade do Porto

Examiner: Professor António Rito Silva
Instituto Superior Técnico, Universidade de Lisboa

Supervisor: Professor Ademar Aguiar
Faculdade de Engenharia da Universidade do Porto

July 15, 2025

Abstract

At the core of every design decision in software architecture is a question, yet they are often overlooked, undocumented, and lost over time. Natural questions are essential to shaping key architectural decisions and preserving architectural knowledge. They arise organically from the architect's experience and the unique characteristics of the system under design. However, natural questions are often mismanaged or ignored, leading to architectural drift, knowledge loss, inefficient resource use, or poor understandability of the system's architecture. This thesis explores how explicitly capturing, classifying, and managing architectural questions can support more structured and collaborative software design. We aim to understand and outline the lifecycle of natural questions, the roles involved, their key requirements, challenges and then envision a specialised tool to support them appropriately.

Based on existing literature, an exploratory case study, a requirements workshop with expert software architects, and following a Design Science Research methodology, we proposed a lifecycle for natural questions. We elicited essential functional and non-functional requirements to develop and prototype a dedicated tool to capture, manage and preserve natural questions. This dissertation presents a prototype tool that assists each role in tracking the lifecycle of natural questions, leveraging generative AI capabilities to provide automated classification of questions, similarity detection and AI-assisted discussion to support collaborative decision-making and architectural reasoning. The tool organises questions in a backlog, preserving rationale, fostering traceability and enabling easy access to architectural decisions throughout the lifecycle.

Evaluating the tool's usability and perceived value is paramount. To do so, a controlled user study was conducted involving university students and experienced software professionals. Participants reported that the tool was intuitive and appreciated the features available, believing that the AI support helped structure reasoning and explore outcomes. However, its influence on decision-making varied depending on user roles.

Ultimately, the results suggested that explicitly managing these natural questions can enhance architectural thinking, foster collaboration and preserve critical design knowledge, laying the foundation for more thoughtful and sustainable software design practices.

Keywords: Natural Questions, Software Architecture, Generative AI, Architectural Knowledge Preservation

Resumo

No cerne de cada decisão de design em arquitetura de software está uma questão, que no entanto, muitas vezes é ignorada, mal documentada e acaba por se perder com o tempo. As questões naturais são essenciais para moldar decisões arquiteturais importantes e preservar o conhecimento arquitetural. Estas surgem de forma orgânica a partir da experiência do arquiteto e das características únicas do sistema em desenvolvimento. Contudo, estas questões são frequentemente mal geridas ou ignoradas, o que pode levar à degradação gradual da arquitetura, perda de conhecimento, uso ineficiente de recursos ou dificuldade em compreender a arquitetura de um sistema. Esta dissertação explora como capturar, classificar e gerir explicitamente estas perguntas pode promover um processo de design de software mais estruturado e colaborativo. O objetivo é compreender o ciclo de vida das questões naturais, os papéis envolvidos, os principais requisitos e desafios, e conceber uma ferramenta especializada para lhes dar o suporte adequado.

Com base na literatura existente, num caso de estudo exploratório, num workshop de requisitos com arquitetos de software experientes e seguindo uma metodologia de Design Science Research, foi proposto um ciclo de vida para as questões naturais. Foram recolhidos os requisitos funcionais e não funcionais essenciais para desenvolver e prototipar uma ferramenta dedicada a capturar, gerir e preservar essas questões. Esta dissertação apresenta um protótipo que apoia cada papel na gestão do ciclo de vida das questões naturais, utilizando capacidades de inteligência artificial generativa para realizar a classificação automática das perguntas, deteção de similaridade e discussão assistida por IA para promover decisões colaborativas e o raciocínio arquitetural. A ferramenta organiza as perguntas num backlog, preservando o raciocínio por trás das decisões, promovendo a manutenção do histórico e permitindo o acesso fácil às decisões ao longo do ciclo de vida.

A avaliação da usabilidade e valor atribuído da ferramenta foi fundamental. Para tal, foi realizado um estudo controlado com estudantes universitários e profissionais experientes em desenvolvimento de software. Os participantes consideraram a ferramenta intuitiva e valorizaram as funcionalidades disponíveis, afirmando que o apoio da IA ajudou a estruturar o raciocínio e a explorar possíveis desfechos e impactos futuros. No entanto, o impacto da ferramenta nas tomadas de decisão variou consoante o papel assumido pelos utilizadores.

Por fim, os resultados sugerem que gerir explicitamente estas perguntas naturais pode melhorar o raciocínio arquitetural, promover a colaboração e contribuir para a preservação do conhecimento crítico de design, promovendo uma abordagem mais estruturada e sustentável no design de software

Palavras-chave: Questões Naturais, Arquitetura de Software, IA Generativa, Preservação do Conhecimento Arquitetural

UN Sustainable Development Goals

The United Nations Sustainable Development Goals (SDGs) provide a global framework to achieve a better and more sustainable future for all. It includes 17 goals to address the world's most pressing challenges, including poverty, inequality, climate change, environmental degradation, peace, and justice [40]. In this section, we focus on the three SDGs that this thesis mainly contributes to.

The specific Sustainable Development Goals mentioned have the following names:

SDG 4 Ensure inclusive and equitable quality education and promote lifelong learning opportunities for all.

SDG 9 Build resilient infrastructure, promote inclusive and sustainable industrialisation and foster innovation.

SDG 17 Strengthen the means of implementation and revitalise the global partnership for sustainable development.

SDG	Target	Contribution	Performance Indicators and Metrics
4	4.4	The tool supports skill development by helping software professionals engage in structured architectural thinking and decision-making.	Percentage of users reporting improved design reasoning; Number of junior engineers and students involved in using or testing the tool.
	4.7	Encourages critical thinking, collaboration, and reflective learning in software architecture, aligning with educational goals.	Inclusion of tool in educational/-workshop settings; Positive feedback on learning outcomes.
9	9.5	Promotes innovation in architectural knowledge management using AI to classify and retrieve design questions.	Multiple AI features implemented; Expert validation and acceptance metrics.
	9.c	Enhances access to architectural design support systems, especially in remote or distributed teams.	Potential for adoption in geographically distributed teams; Tool designed for asynchronous, location-independent use.
17	17.6	Fosters collaborations and knowledge sharing across organisations through structured question-driven design archives.	Evidence of cross-organisational collaboration; Frequency of shared use cases or contributions.

Acknowledgements

As I finish this important chapter in my academic journey, I find it essential to express my sincere gratitude to the individuals whose support has been invaluable in the successful completion of this thesis.

To FEUP, for being the cornerstone of my academic path. The knowledge, learning experiences, and growth I have gained during my time here have shaped both my academic and personal development. I am truly thankful for the opportunities and guidance provided throughout my studies.

To Professor Ademar Aguiar, my thesis supervisor, thank you for your invaluable mentorship, constant support and for the countless meetings where your insights, feedback, and encouragement helped guide this thesis to its final form.

To my parents, for your unconditional love, unwavering support, and for nurturing the curiosity that drives me. Your support and motivation have shaped who I am, and I am deeply grateful for everything you've given me.

To my sister, for your steady support and encouragement throughout this journey. I'm grateful to have had you by my side every step of the way and to have you as my longest and enduring friend.

To my grandmother, for your devotion and ongoing support throughout my life. Your strength, kindness, and unwavering belief in me have always guided me. I am forever thankful.

To my godparents, for their unconditional kindness and unwavering support over the course of my journey. I sincerely appreciate your love and guidance.

To Joana, for your endless love, patience, understanding, and for being my best company through every challenge and success. I'm deeply grateful to have you by my side.

To my friends, thank you for your constant encouragement, always making me laugh and for the many memories we shared. Your company has brought joy and motivation along the way.

Lastly, to everyone who contributed in any way to the completion of this work, whether directly or indirectly, I extend my heartfelt thanks. Your support has made this accomplishment possible.

Diogo Lemos

"It's the job that's never started
as takes longest to finish."

J.R.R. Tolkien

Contents

List of Figures

List of Tables

List of Acronyms

SDG	Sustainable Development Goals
AK	Architectural Knowledge
LLM	Large Language Model
ADR	Architectural Decision Record
NLP	Natural Language Processing
AI	Artificial Intelligence
GenAI	Generative Artificial Intelligence
RAG	Retrieval-Augmented Generation
FR	Functional Requirement
NFR	Non-Functional Requirement

Chapter 1

Introduction

In the process of software architecture design, from the requisites to the architecture, questions emerge that are disregarded and mismanaged irrespective of the importance they hold in shaping key decisions, often leading to incomplete knowledge and sub-optimal future-impacting decisions.

1.1 Context and Motivation

Throughout the design and development of software architecture, natural questions organically emerge during the entire length of the process. As stakeholders and architects aim to solve the complexity and uncertainties within the system, spontaneous, context-driven questions come up. These questions are of highest importance since they frequently expose underlying issues, dependencies or areas of uncertainty that may eventually lead to inefficient and ineffective decisions[19]

It is crucial for these questions that arise to be answered in a systematic and timely way. These questions and answers are of utmost importance since they can involve both the problem and solution space, or the management phase, among many others. Questions may eventually receive a temporary response established on assumptions, depending on when they are answered [16].

This thesis refers to these naturally occurring questions as "natural questions". Natural questions arise organically and spontaneously during the architectural design process as members reason and discuss through complex tasks, trade-offs, unknowns or assumptions [16]. Unlike structured requirements, natural questions are context-driven, evolving with the situation and the overall systems, which often help surface latent or unrevealed architectural concerns. These may emerge in any situation, ranging from meetings, discussions, reviews and informal conversations with others, and they play a crucial role in shaping architectural decisions. Due to their dynamic and unstructured nature, they are challenging to track and preserve using conventional tools [4].

The importance of these natural questions is very high, impacting the preservation of architectural design knowledge and their respective answers(definitive or temporary), which can be essential in the moment and the latter phases. To achieve this requirement, architects must be supported with specialised tracking tools, accessible throughout the questions' lifecycle, from their inception to their eventual conclusion and archival, given the nature of constant changes and loss

of information. Neglecting these architectural questions, although very frequent, is not confined to the design phase of software development [7]. It continually extends its impact through the overall integrity of every project since it affects the maintenance, scaling and system redesign. As per Salehin [32], mistakes introduced during the requisites stage can cost upwards of three times to repair if they are found during the architecture phase and five to ten times more costly when found in the development phase. The nature of questions is dynamic and unpredictable, and these often occur after the initial system architecture, evolving alongside the system to meet the changing requirements, technological advancements or obstacles.

This thesis investigates how to support the eliciting, organising, and tracking of architectural questions. A plethora of GenAI technologies will be explored and applied to support multiple dimensions of organising all the questions and understanding their context, with the timing and schedule of questions to be addressed and stored to progress on the architecture.

GenAI solutions are being increasingly explored in many software-related areas, from initial design and development to ongoing maintenance and support [31]. Most of these developments and implementations are related to the upkeep and writing of code, such as refactoring tools and AI assistants that can understand a particular necessity explained in a common language by a developer and generate code for it [26]. However, as stated before, these developments are leaving a gap in software architecture.

1.2 Problem Identification

The lifecycle of software architecture questions is indispensable in developing any project, even though it has been ill-managed numerous times. Each question's lifecycle holds more information than just its response and an understanding of the project as a whole. Therefore, gathering and storing as much of each question's iteration is paramount to maintaining easily accessible knowledge [16].

Given the importance of such a task, several difficulties arise since the nature of questions is not always easily tracked in text and stored efficiently. Some traditional methods, namely post-its, chat messages and disorganised records, are unequipped to be stored as neatly as they should, given their importance. Also, these traditional methods fail to preserve the context and rationale surrounding a question [17].

Questions are not static and arise in numerous ways that must be addressed and preserved. These can take shape in written conversations, discussed via speech and shared through diagrams and images, making storing all the knowledge demanding and laborious [32]. Speech-to-text recognition is vital for question iteration, as numerous decisions can be made during a conversation. Capturing insights and questions through an easily accessible speech-to-text-powered app would allow users to input questions effortlessly, removing one of the primary sources of friction in documenting spontaneous architectural thoughts.

Considering these methods and exploring GenAI and LLMs capabilities, finding a way to store information gathered from text, speech, and image is necessary. As a first phase, there is a need

to comprehend and research a more suitable way to store and preserve these pieces of information and subsequently explore the LLMs' capabilities to understand questions and be able to assist in answering questions related to the software architecture, ranging from questions related to the design to questions regarding decisions that require reminding. Generative AI is crucial due to the adaptability it provides by analysing and updating the knowledge repository in a structured and constant way, being exceptionally suited for the dynamic nature of questions [2]. Comprehending the nature of software architecture, there is also a difficulty that arises from configuring an LLM for it to understand and be able to reply to any question that may surface for every architecture [29].

1.3 Research Questions

To investigate the potential of GenAI technologies for preserving and grasping knowledge derived from questions related to software architecture, the most prominent research questions addressed in this thesis are as follows:

RQ1. Are the traditional methods insufficient for retaining knowledge conveniently in all phases, thereby justifying a need to integrate GenAI technologies?

RQ2. How successfully can GenAI technologies be used to anticipate the repercussions of each decision and aid the process of decision-making to be more effective?

RQ3. To what extent can a specialised tool contribute to structuring the management of architectural questions throughout the software design process?

1.4 Hypothesis

Considering the previously identified challenge, the hypothesis that we will investigate and evaluate is as follows:

> "An optimally configured set of AI-enabled tools will make it more precise and efficient the tracking, preservation and retrieval of questions and respective knowledge when compared to traditional methods"

1.5 Dissertation Structure

This dissertation is organised into six chapters, each building upon the previous to comprehensively study the feasibility of managing natural questions in software architecture design. The structure is as follows:

- **Introduction** This chapter provides the context and motivation behind the dissertation, introduces the main objectives and outlines the structure of this thesis.

- **Background and related works** This chapter reviews the existing literature on software architectural design, AI approaches to knowledge preservation and the role of questions in design processes. It identifies key insights, challenges and limitations in current practices and tools, providing the theoretical foundation for the research.
- **Foundations of the Proposed Solution** This chapter introduces the methodological and conceptual foundations of the proposed approach to preserve and manage natural questions. It outlines the lifecycle of questions in architectural decision-making, describes the requirements elicited through an exploration case study and a survey with experts and reframes the problem within current architectural challenges.
- **System Architecture and Prototyping** This chapter details the design and implementation of the prototype system. It describes the system architecture, prototypes, key components, user interface, and AI technology integration to support question classification, similarity detection, and discussion tracking.
- **Evaluation** This chapter describes the evaluation methods used to assess the feasibility and perceived value of the prototype. It describes the findings from a controlled experiment with experts and university students, analysing how well the tool supports architectural knowledge preservation and how useful it would be in a professional work context.
- **Conclusions** This final chapter summarises the key findings of the dissertation, outlines the main conclusions, and suggests areas for future research by addressing the limitations of this investigation.

Chapter 2

Background and related works

2.1 Literature Review

A substantial body of relevant literature is one key aspect that contributes significantly to the thoroughness of a dissertation. In-depth research of existing literature and research leads to a better understanding of the foundation to build upon in this thesis, aiding the process of answering and complying with the questions and targets set previously.

The method chosen for this literature review was a systematic review. This procedure was selected based on analysing several critical parameters: speed, reproducibility, comprehensibility, and bias. A systematic review can be described as a structured and transparent process that identifies and synthesises applicable literature. It ensures that the results are reliable and effortlessly understandable while minimising the potential for bias in selecting and evaluating available research. By adhering to the PRISMA 2020 guidelines, this systematic review can improve the transparency and thoroughness of the research process.

2.1.1 Search Strategy

In this method, retrieving relevant academic documents and research was crucial. To aid in this task, a search query, highlighting keywords in this research, was developed to explore several academic publication databases, given their nature of providing access to a wide range of knowledgeable and relevant documents.

Academic Databases and Sources

A fundamental part of the search strategy revolves around deciding which academic databases to select for this process. The four academic databases selected were **ACM Digital Library**, **IEEE Xplore**, **Scopus** and **Science@Direct**. It is essential to know that some query-related restrictions from **Science@Direct** required a downsampling of the query that will be presented in this section.

Keywords

Throughout the initial phase of this research, several words were captured and seen as fundamental and a close approximation to the themes explored, as well as being significant to the research questions previously mentioned. Given the nature of several concepts' varying nomenclature, synonyms, and meaningful but different interpretations of each word, they were also identified for each keyword. The three keywords defined were as follows.

Software Architecture - "microservice architecture", "software architectural decisions", "software architecture knowledge base", "software design decisions", "Architectural Knowledge".

Generative AI - "LLM", "GenAI", "large language models", "language model", "AI-driven tools"

Knowledge Tracking - "knowledge management", "knowledge elicitation", "knowledge preservation", "question tracking".

Search Query

Following the understanding of the importance of these keywords and their synonyms, the process of defining a search query to be used on the selected databases was initiated, guaranteeing that it effectively captures pertinent literature and aligns with the research objectives. Using the boolean operators **AND** and **OR**, the keywords could now be connected to find relevant research documents. After several phases of testing to find an ideal query, the one that presented the subsequent results is as follows. The keywords were connected to their synonyms by **OR** operators and connected between them through the **AND** operator.

("generative AI" OR "LLM" OR "GenAI" OR "large language models" OR "language model" OR "AI-driven tools") AND ("Software Architecture" OR "microservice architecture" OR "software architectural decisions" OR "software architecture knowledge base" OR "software design decisions" OR "Architectural Knowledge") AND ("knowledge management" OR "knowledge tracking" OR "knowledge elicitation" OR "knowledge preservation" OR "question tracking")

As stated earlier, **ScienceDirect** required a modification of the query to make it more brief to comply with the platform's constraints. The query used on this academic database was:

("generative AI" OR "LLM") AND ("Software Architecture" OR "software architectural decisions" OR "software design decisions") AND ("knowledge management" OR "knowledge tracking" OR "knowledge elicitation" OR "knowledge preservation" OR "question tracking")

Publication Retrieval

This process resulted in the retrieval of **344** publications. The following table illustrates the results retrieved from each academic database.

Database	Count
ACM Digital Library	134
IEEE Xplore	50
Science@Direct	82
Scopus	78

Table 2.1: Number of Publications from each Database

ACM Digital Library was the academic database that presented the most results, with 134, followed by **Science@Direct** and **Scopus** with 82 and 78, respectively. Lastly, **IEEE Xplore** yielded the lowest amount of documents, with 50; this may have resulted from searching only within the abstract of each article.

2.1.2 Selection Strategy

Inclusion and Exclusion Criteria

Phase	Inclusion/Exclusion criteria
1	Search engine results after executing a search query
2	Date of the publication from 2018 to 2024
3	Full-text access
4	Published in the English language
5	Published in the area of “Software Architecture” or “Generative AI”, "Large Language Models", "Microservice Architecture", "Architectural Knowledge", "Knowledge Preservation" or "Question Tracking"
6	Not in the field of IoT
7	Not duplicated
8	Not retracted publication
9	Not work-in-progress publication
10	Article, Conference Paper or Book
11	Not summaries of workshops, seminars, lectures, or tutorials

Table 2.2: Inclusion and Exclusion Criteria

Screening Phase

The screening process can be described as one of the most crucial parts of every literature review. After the initial retrieval of the results, the first phase was to remove the duplicates. Utilising tools such as 'Parciv.al' [39] and 'Rayyan' [28] facilitated the process of removing duplicates and the overall screening of each retrieved document.

From the initial **344** publications, **50** were duplicated and removed from the following stages, reducing the number of articles to 294.

Given the second inclusion criteria, Date of the publication since 2018 (inclusive), several articles were removed promptly based on their publication date.

Using Rayyan's capabilities, screening an immense number of results was drastically improved by filtering keywords deemed fundamental to this thesis. With the removal of duplicates, exclusion of articles that did not agree with the publication date criteria and the addition of keyword filtering, the number of articles was reduced to **68**.

From these 68 articles, the next step was to skim through as much content as presented by the articles to understand their relevance to this research.

Results

In conclusion, this literature review process began with identifying 344 publications. Following a thorough and rigorous revision and exclusion of articles aligned with the research objectives and deemed as not promising, **19** articles were identified as potentially robust and relevant for the subsequent stages of this study.

2.2 State of the Art

State of the Art is a cornerstone chapter when writing a thesis. In this chapter, the innovative nature of this work is complemented and proven by the extensive research of other academic and literary documents from similar areas, touching on several key aspects deemed vital to the overall thesis. This chapter is greatly complemented by the previous one, Literature Review, since several papers and documents were retrieved in that process. By having a well-formatted and tuned literature review, from the thousands of documents that mentioned keywords deemed necessary, selecting those crucial to the development of this thesis became more manageable and facilitated a smoother selection.

In addition to the documents retrieved from the method described previously, a second phase of documents was deemed interesting and potentially crucial to this thesis. These consisted of documents that were recommended and articles that were quoted during the search and reading process. A method was applied to classify each paper retrieved in the literature review, helping to understand which documents were the most promising to the overall research. By skimming and analysing each of these, from the initial list of 19 papers, six were deemed extremely important to

the overall research, and the rest were regarded as interesting, even though somewhat less focused on the required area than the main 6.

2.2.1 Background

The process of software architecture design is driven by the systematic questioning of both the problem and solution space, which shapes the final system. These questions are necessary, and their unpredictable pattern reflects how humans develop ideas or refute them. This process is not linear, and every question holds a meaning that shapes the overall project. The main issue with this non-linear process is that questions that have meaning and knowledge can be forgotten or dismissed in an initial phase, while being important later on. The nature and importance of these questions are dynamic and evolving with the project. These questions can be answered provisionally with assumptions, even when not answered definitively. When not tracked correctly, these assumptions can be forgotten and cause significant issues in the long run [41]. Neglecting these questions may lead to considerable risks, such as misaligning project objectives, inconsistent design decisions or even dependency conflicts. By failing to manage and preserve these questions and consequent answers or assumptions, teams may overlook underlying issues that can manifest in later stages, becoming increasingly challenging to address and resolve effectively.

The importance of these natural questions is not contested, yet their tracking is loosely managed due to the lack of accessible and easy-to-use solutions. In a study related to these questions that arise during software architecture design [16], the tracking and organisation of natural questions is multiple dimensional with the often use of ad hoc methods, with some mentions of platforms, such as *Jira* and *Trello*, and informal methods, for example, to-do lists. While not specialised for architectural questions, these methods are also the probable cause of the lack of preservation of architectural knowledge. To address these issues, the study recommends the development of a specialised tool that can support the eliciting, organising and tracking of natural questions in software architectural processes.

Complementary to knowledge preservation, the systematic tracking of questions also stimulates critical thinking and provides valuable context for past decisions. By emphasising the role of questions in leading architectural decisions, this paper highlights the importance of correctly managing these inquiries to avoid the risks associated with assumptions and provide well-established software architecture projects. Furthermore, by managing these questions methodically and organised, collaboration with others is enhanced, ensuring that everyone involved in the design process understands key insights and assumptions.

The flowchart below represents the evolution of a question through the many stages until its archival.

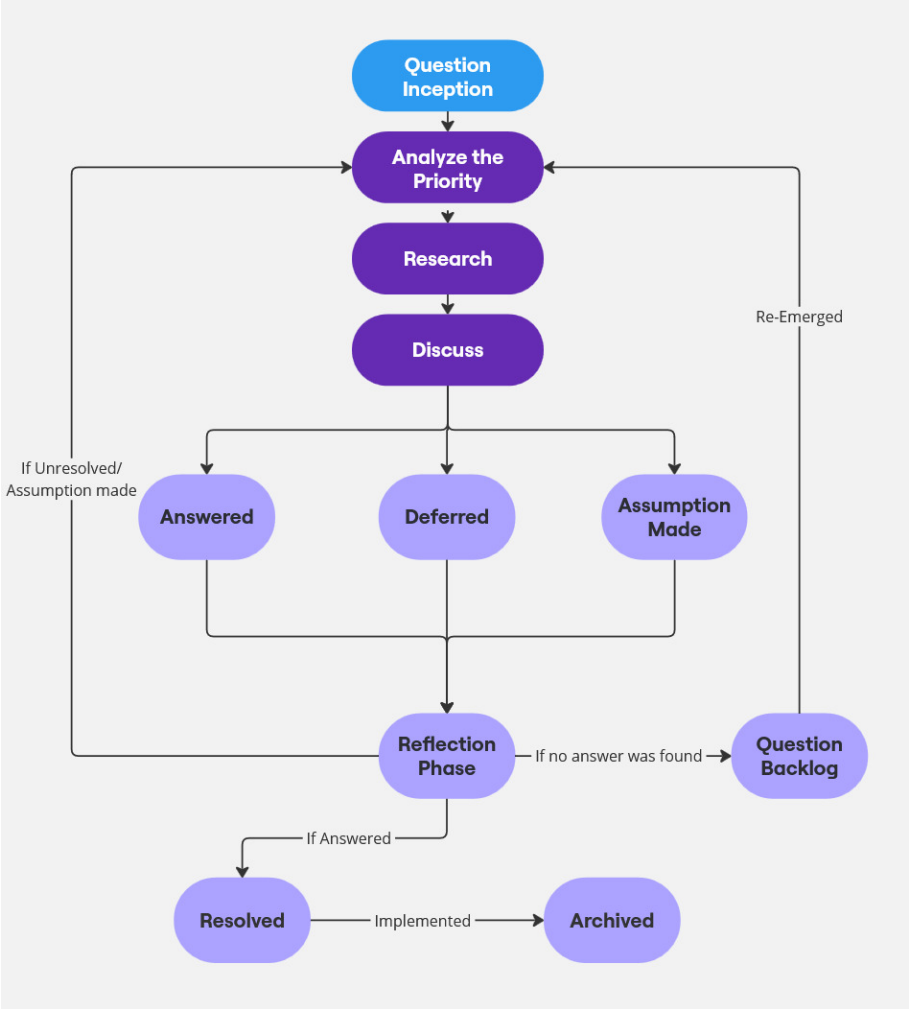


Figure 2.1: Evolution of Natural Questions

2.2.2 Architectural Knowledge Preservation

As software development and software engineering become increasingly intricate, one critical concern for projects and organisations is the preservation of architectural knowledge(AK), since it plays a crucial role in the successful development and maintenance of every software project. AK encompasses design decisions, architectural solutions, patterns and principles that shape the overall structure and evolution of systems [18]. Mismanagement of AK and its loss leads to a risk of losing vital information with overarching consequences due to team changes, technological alterations, and insufficient documentation.[38] Architectural knowledge preservation aims to address these challenges by providing systematic approaches to retrieve, document and organise AK to ensure that valuable information and decisions are not lost over time. Anwar et al. [3] defend that the importance of documenting architectural knowledge is well established among communities of researchers and practitioners since this knowledge evaporates over time if not correctly reported. Preserving essential documentation and decisions enhances the quality of software projects and

the knowledge transfer between members, and eases their maintenance and redesigns throughout their life cycles.

A study exploring the feasibility of using Large Language Models (LLMs) to generate Architectural Decision Records(ADRs)[10] states that these records are essential in the documentation of key decisions in software architecture, which ultimately promotes transparency, collaboration, and understanding. Documentation debt prevention is also necessary, particularly with the proliferation of agile software development and global software engineering[5].

Architectural Knowledge preservation is of utmost importance, and understanding its challenges and insights is key to understanding how it relates to the study of natural questions preservation. Since questions often hold more value than answers, comprehending the lack of knowledge preservation tools is extremely important to this thesis.

Challenges

Architectural knowledge preservation within the realm of software development is paramount, yet it faces several challenges. As software systems grow in complexity and teams are assembled, disassembled and interchanged, ensuring that AK is adequately documented and maintained becomes increasingly complex. The need for accessible use and understanding of AK is essential for team members and stakeholders, and without a practical and accessible approach to preserving AK, vital decisions, ideas and patterns can lead to the stagnation of the evolution of projects. These challenges stem from multiple factors, such as time constraints, multiplatform and informal documentation practices and adversity to the current lacklustre solutions.

The required effort and time for documentation are perceived as a drawback in preserving ADRs. With the fast-paced approach to projects, architects and developers often face tight deadlines and end up neglecting documentation to deliver functional software features or bug fixes. This oversight of documentation leads to inconsistent and usually outdated records. As noted in the study by Dhar et al. [10], architectural decision records(ADRs) are essential for tracking design decisions. Still, they are neglected since it is seen as disrupting the workflow in the development phase. This time constraint is a challenge that must be addressed for a solution to be accepted and standardised.

Another significant challenge to preserving AK is the informal documentation practices and the multiplatform use of these. The AK gets fragmented since it can often spread across several informal channels such as messages, emails and informal meetings.[12] The lack of a tool that would make the documentation standardised and centralised is the bane of the difficulty of systematically capturing all the relevant decisions, ideas and patterns. The lack of an appropriate solution would negatively affect the way teams and, more concretely, new members locate essential information in a swift and accessible way. This would emphasise the knowledge gap between current and newer members and lead to redundant and conflicting work. According to the study by Dhar et al. [11], this issue exists because most of the existing AKM systems are manual and deterministic.

Despite the extended list of advantages of preserving AK, there is frequent resistance to adopting formal documentation tools. As stated before, with the proliferation of agile development practices, many teams prefer informal, flexible methods that prioritise swiftness and direct communication, neglecting detailed documentation. This explains the lack of adherence to formal tools, which are viewed as counterproductive and clunky. This is described in studies regarding collaborative software engineering[5], where developers often rely on messages, emails or informal diagrams instead of adopting specialised AKM tools.

Lastly, a big challenge inferred from multiple papers revolves around tacit knowledge and how difficult it is to transfer it. This knowledge is held by individuals based on experience and context, and capturing it is particularly challenging since it can not be easily quantifiable and documented. In the absence of proper tools or processes, this type of knowledge remains undocumented and can be easily misunderstood, leading to shortages of knowledge in vital parts of the project over time.[5]

Insights

Recent studies highlight the elements necessary for practical architectural knowledge preservation, addressing some of the previously described challenges.

One key insight is related to centralised repositories. Centralised repositories are critical for organising and storing AK since they ensure accessibility, consistency and traceability of decisions and ideas across teams and projects. As mentioned in the study by Dhar et al. [11], consolidating information facilitates collaboration and helps teams sustain a shared understanding of the architecture and decisions made.

It is stated that an approach to use real-time documentation tools, such as interactive bots [12], would help capture architectural decisions as soon as they occur. This would minimise the risk of knowledge loss and, with ease, would record decisions in semi-structured templates during team talks or even agile sprints, ensuring accuracy and even reducing the burden of documenting everything after each meeting [23]. In the example of this paper by Gilson and Weyns [12], interactive bots could generate summaries for immediate validation, bridging the documentation to the actual decision-making process.

Integrating architectural knowledge management systems with existing development tools is believed to make it easier for teams to adopt these systems [42]. One example of a development tool that would fit this goal is a version control system [12]. Embedding AK capture into the teams' day-to-day workflows would ensure that documentation becomes a natural part of the development process.

ARC3N, a collaborative uncertainty catalogue that organises the recording of uncertainty in architectural decision-making proposed by Hahner [14] stresses the significance of structured, collaborative knowledge bases in software architecture. This aligns with the overarching objective of this thesis, which is to systematically capture and manage natural questions to surface hidden uncertainties during design.

Now more than ever, generative AI is essential to software engineering's present and future. Utilising generative AI and information extraction tools would reduce the manual effort required and ensure that the AK is preserved by automating the creation of ADRs and other documentation [10]. This would be crucial when time and resources are constrained [11]. AI can also preserve knowledge across different languages since it can accurately translate documents and texts while recognising images, which is key for maintaining important artefacts and documents [15].

There is an incentive to focus on usability and adoption from teams and stakeholders. The priority should be making the architectural knowledge management tools as flexible and functional as possible, encouraging wide-range adoption [11]. Systems should have their interfaces be user-friendly and tailored to different roles, such as architects, developers and project managers, enhancing their usability and prompting and encouraging consistent documentation practices [12].

The following section will refer to many approaches regarding AI usage, which will be discussed in the next section, AI Approaches to Knowledge Preservation. Focusing on a unique approach proposed in the paper by Borrego et al. [5], using NLP-based tagging systems was seen as beneficial to classify AK within agile teams. By recommending and integrating tags for communication in tools like Slack, these tagging systems would streamline the organisation and capture of AK.

2.2.3 Collaborative Knowledge Creation: The Ba Concept

The Ba concept, presented by Ikujiro Nonaka and Noboru Konno [24], provides a framework to understand how knowledge is created, shared and assimilated within a collaborative environment. "Ba" is considered a shared space for emerging relationships. Aligning with the SECI model of knowledge conversion — Socialisation, Externalisation, Combination, and Internalisation — the Ba concept provides a more dynamic view on how these transformations happen within collaborative spaces and also defines four modes of knowledge conversion, which are:

- **Originating Ba:** Knowledge is shared through direct interactions, as when discussing questions between architects.
- **Interacting Ba:** Knowledge is articulated into explicit knowledge, as when documenting and structuring questions
- **Cyber Ba:** Knowledge is organised, linked and enhanced through tools, aligning with our vision of AI-driven solutions to structure and retrieve architectural questions.
- **Exercising Ba:** Explicit knowledge is reabsorbed into tacit knowledge, allowing architects to learn from past questions and refine future decision-making.

This concept and framework are essential for designing an environment that supports continuous capture, management, and understanding of architectural questions. The Ba model reinforces the idea that questions should not be regarded as something static but dynamic and actively evolving within a structured and interactive environment to facilitate long-term knowledge preservation.

2.2.4 AI Approaches to Knowledge Preservation

AI-driven approaches can transform the entire landscape of how architectural knowledge is documented, organised and retrieved. Unlike the traditional manual methods, neglected at times by being time-consuming and prone to inconsistency and incompleteness, AI offers an opportunity to reduce these current challenges by providing automation, adaptability and scalability [20]. Through an extensive range of leveraging techniques such as Natural Language Processing(NLP) and machine learning, AI can extract, generate and classify architectural knowledge from numerous sources [34]. These processes would ensure that vital decisions and ideas are not lost over time and are preserved accordingly to be used at any time by any team member or stakeholder.

With the predominance of agile practices, informal communication tools, and the growing number of work-from-home employees, AI approaches have become highly relevant since a lot of knowledge lacks structure and stability, [12]. These methods aim to enhance collaboration, reduce time consumed on documentation and enable teams to focus on different high-value tasks while preserving the accessibility and traceability of architectural decisions [4]. Ethical frameworks and human-in-the-loop methods must guide these advancements to ensure responsible, transparent, and sustainable AI integration. [8, 13] With the ongoing advancements in AI, its ability to handle complex and even fragmented knowledge keeps improving and becomes essential for modern software engineering. Studies firmly believe that collaboration with AI highlights the potential of AI as a partner in development and research, due to its ability to provide architectural insights, [9].

Challenges

Indeed, even though AI approaches significantly help knowledge preservation, they have their drawbacks. Some challenges range from the lack of ability to replicate human interaction to the limitations it might have currently that could be addressed in the future.

Starting with the study from Dhar et al. [10], in the case of AI-generated knowledge, such as architectural decision records, it would often be devoid of depth and contextual relevance compared to manually created ones. While being able to generate knowledge swiftly and at a reasonably high level nowadays, one of the most pronounced issues still is the inability to replicate and achieve human-level performance in this area fully. A justification for this is stated to be that AI models require high-quality data for fine-tuning, which could be challenging to come across, and generating knowledge from inadequate documentation remains challenging [5]

A main drawback for companies and teams interested in integrating AI solutions with their workflows is the difficulty of combining them with diverse tools and systems. This scalability requires the assurance that the systems will work seamlessly across varied tools and systems. Furthermore, the paper by Gilson et al. [12] highlights that the informal nature of decision-making in software architecture and the resistance to adopting formal tools would hamper the integration process.

Complementary to the challenge stated before, there is also an obstacle that must be addressed: resistance to change. Parallel to traditional architecture knowledge preservation challenges, teams may resist adopting AI-driven systems due to several factors. These could be unfamiliarity, complexity or apprehensions towards disrupting existing workflows [5]. Also, it is believed that applying this approach to large and complex systems would require a significant amount of resources, as stated in a paper by Rukmono et al. [30].

When reflecting on the use of AI, more specifically, cloud-based solutions, while powerful, concerns are raised regarding privacy and security. The storage and processing of vital and private architectural knowledge may be at risk, and to reduce it, many organisations resort to local, secure solutions, which, as well as being resource-reliant, can also limit the adoption of AI.

Insights

Several insights and approaches were captured, and many papers and articles were explored. With their knowledge, one can understand what AI approaches might be the better fit for knowledge preservation.

A key area in which AI is remarkable is the automation of repetitive tasks [20]. Using this insight, AI is recommended to summarise discussions or generate ADR, allowing teams to focus on more creative or specialised areas of software engineering. This study also researched a variety of LLMs using zero-shot, few-shot, and fine-tuning approaches to ensure relevance and accuracy. Smaller AI models offered a balance between affordability and performance by being able to be deployed locally. This is the case of the model researched by Dhar et al., the Flan-T5, which was described to achieve comparable results after fine-tuning to larger models [10]. By deploying models on local servers, the challenge related to privacy concerns is then addressed, causing AI approaches to be more accessible to organisations with strict data policies [11].

Collaboration is heightened with the use of AI approaches. AI tools, bots, and assistants can offer real-time monitoring, intervention, and documentation during discussions and meetings, capturing the architectural decisions and presenting the knowledge captured for review and validation. This approach reduces the risk of lost knowledge and highly improves the documentation quality, aligning it with current decisions [12]. Advanced systems compile knowledge from various data types, such as documents, code and diagrams. Through retrieval-augmented generation(RAG) techniques, there is an enhancement to the generation of AK by combining retrieved information with generative models [11].

Natural language processing(NLP) systems improve the organisation and retrieval of knowledge efficiently, reducing the workload on team members. In the study "Tags' Recommender to Classify Architectural Knowledge Applying Language Models" [5], an NLP-based system is used to classify and tag knowledge from unstructured sources, ensuring the rapid location of relevant information during development. These sources, for instance, emails, messages or Slack messages, are packed with content and knowledge, and an approach for capturing and uniting it would allow for the preservation of important information, especially when leveraging semantic tools to ensure interoperability across platforms [33]. A tailored AI system can provide specific insights and

transmit knowledge based on the user's role and unique needs, raising the adoption rate that would otherwise be relatively low. Generative AI is also critical and incentivised by small and medium enterprises since it allows these companies to close the knowledge and technological gaps with larger ones [27].

AD-Caspar is an example of how AI must be incorporated into the tool. AD-Caspar is a chatbot illustrating how cognitive AI agents can interpret and respond to natural questions using reasoning [21]. This is a capability that, if leveraged with the tool, would improve architectural decision discussions exponentially.

2.2.5 Questions in Software Architecture and Knowledge Preservation

The role of questions in Software Architecture and Knowledge preservation may go unnoticed at times. However, every process is inherently driven by the questions arising from the problem and solution domains. These questions shape every system while also being a core part of decision-making and being able to clarify and elaborate on assumptions. This dynamic questioning process mimics the nature of problem-solving, where questions drive and evolve alongside the project.

Questions tend to be perceived as extremely valuable only when answered. However, questions hold significant value even when they can not be answered or temporarily settled through assumptions [19]. This is due to them manifesting a key consideration or challenge faced during a specific project point. Properly documenting these questions ensures that the context and reasoning behind architectural decisions are preserved, aligning with the goal of traceability and effortless transfer of knowledge between members [25]. Despite their importance and role, these questions are being managed loosely, carelessly, often utilising ad-hoc methods or general-purpose tools, which are not specialised for architectural needs.

Challenges

The main challenge opposing the preservation of questions is related to how they are managed and recorded. These questions are scattered across multiple communication platforms and tools, as exposed by Borrego et al. [5], demonstrating the difficulty in preserving vital questions and answers systematically. As stated, questions answered provisionally or not answered are just as crucial as resolved ones. However, these are often not documented or tracked, increasing the potential for knowledge loss and forgotten reasoning, introducing risks in the later stages of development [18]. The paper "Can LLMs Generate Architectural Design Decisions" [10] highlights the risks associated with incomplete or undocumented decisions.

Even when teams try to document this knowledge, it is primarily through informal methods such as to-do lists, which are insufficient to capture and manage architectural decisions effectively and often fail to capture the evolving nature of these questions [16]. This is due to a lack of specialised tools tailored to track architectural decisions, hampering the ability to preserve these questions methodically. The dynamic nature of questions is captured through static methods, lacking the flexibility to adapt to evolving knowledge needs. Questions are also vital in ensuring

effective communication between team members and AI. The constantly transforming landscape of software development posed a challenge in drafting relevant questions since it is necessary to update them to reflect the latest advancements and best practices [9].

Insights

Several insights were inferred from various papers and articles relating to the role of questions in these domains.

Firstly, it is believed that multimodal AI systems can consolidate diverse knowledge sources, allowing for the capture of the questions that arise in each context. These questions can be gathered from codebases, documentation and even discussions, being a priority to create centralised repositories for ensuring the preservation of every knowledge and question, supporting the dynamic and, at some points, fragmented nature of queries and question-driven knowledge [11]. Dhar et al. also interpret in another paper that architectural decision records often stem from answering critical questions and aid in tracking and preserving these inquiries [10].

A few papers briefly mention or infer the role of questions in software architecture, focusing on natural language processing(NLP). It is acknowledged that interactive NLP tools could capture and summarise the transformative cycle of questions in real time during meetings and discussions. This would be a preponderant factor in ensuring that architectural questions are documented as soon as they materialise, reducing the risk of knowledge loss [12]. As mentioned, a tagging system proposed by Borrego et al. would also be key in classifying and retrieving architectural knowledge through questions, supporting better decision-making and cooperation [5]. Questions play a crucial part in AI since through them we can extract valuable insights, and having this collaborative questioning helps in understanding diverse methodologies and adapting them to each project [9].

2.2.6 Current Tools used to Manage Architectural Questions

As discussed previously, as well as research being done toward the applicability and feasibility of a tool to support the management of architectural questions, the focus on researching the current tools that are used to manage these questions is significant to understand what deficiencies and what is there to improve on existing solutions.

In the following table 2.3, existing tools and platforms commonly used to capture and manage information during software architecture design are presented through a comparative overview, highlighting their capabilities and limitations. While platforms like Github Issues and even Trello or Jira offer partial support for question tracking through issue or task annotations, they lack specialisation and explicit mechanisms to capture architectural rationale and reflective thinking. Miro and the likes of Mural are, in comparison, more suitable for collaborative brainstorming, but fail to provide built-in support for temporal tracking or even structured reflection. Notion is a good option due to its flexibility and capability of capturing information manually. In contrast, it requires a significant setup effort and is far from accomplishing architecture-specific tasks and providing

proper support. Slack and Teams, becoming increasingly popular among working environments, support the collaboration intended for the ideal tool, but fall short in tracking the evolution of questions and maintaining an organised temporal context. Finally, as stated before, informal and ad-hoc methods such as post-its and to-do lists are still valued for their ease of use and visibility in determined settings, yet as for informal and manual processes, they pose significant limitations in terms of searchability, traceability and especially long-term preservation[4]. Overall, in the current tools, there does not seem to exist a tool that supports the lifecycle of naturally occurring architectural questions to its full extent, particularly in terms of tracking, context, and reflection, which is exceptionally compelling in the realm of questions. This highlighted gap further supports the need for a dedicated solution to manage these naturally occurring questions in software architecture contexts.

Tool / Platform	**Question Tracking**	**Temporal Context**	**Collaboration**	**Reflection**	**Arch-Specific**
GitHub Issues	Partial	Yes	Yes	No	No
Trello / Jira	Partial	Yes	Yes	No	No
Miro / Mural	Partial	No	Yes	No	No
Notion	Yes (manual)	Yes	Yes	No	No
Slack / Teams	No	Partial	Yes	No	No
Post-Its	No	No	Yes	No	No

Table 2.3: Comparison of Tools Commonly Used to Manage Architectural Questions

Chapter 3

Foundations of the Proposed Solution

This chapter lays the foundation for the proposed solution to support the capture, management, and preservation of natural questions during software architecture design. Building upon the theoretical insights and practical needs identified previously, this chapter complements the knowledge gathered by an exploratory case study with students in the area and expert feedback from knowledgeable software architects and engineers. These observations allow for gathering and refining the requirements and validating the core concepts, ensuring that the proposed solution is grounded in both academic understanding and real-world practices. This chapter concludes with the architectural design and rationale behind the developed prototype, which embodies the lifecycle of natural questions and demonstrates its feasibility and relevance.

3.1 Design Science Research

The development and evaluation of our tool for successfully managing and preserving architectural questions relies on the Design Science Research methodology. The DSR method focuses on creating and assessing artefacts that address encountered challenges through iterative refinement and validation. DSR is ideal for this type of research since it aims to create and evaluate artefacts that solve identified problems in a domain-specific context [6].

This methodology provides a framework for achieving the goal of this thesis by combining knowledge from the literature review, empirical exploration, and iterative artefact design. Ensuring the solution is relevant and rigorous throughout the development and evaluation phase is key.

This process involves:

- Identifying challenges from the comprehensive review of the literature and an exploratory study (including workshops and surveys) to uncover the key challenges architects face when managing natural questions in software architecture.
- Eliciting requirements based on the identified challenges and expert input, serving as a foundation for artefact design and iterative development

- Designing and prototyping an initial architecture and mock-up of the tool, including low-fidelity prototypes evolving into high-fidelity ones by empirical feedback and theoretical grounding.
- Developing a functional implementation using an iterative approach, testing and demonstrating it in realistic architectural contexts.
- Evaluating the tool against its design objectives through qualitative and quantitative methods, assessing whether it provides practical value and theoretical contribution.
- Reflecting on the whole process is pivotal, aiming to improve further the tool to maximise its usability, performance, and real-world applicability.

3.2 Exploratory Case Study

Understanding the requirements for a solution to track and preserve knowledge from natural questions is fundamental, laying the groundwork for future research and development. To elicit and validate requirements, an exploratory case study was conducted within the scope of a Master's course on Software Systems Architecture at the University of Porto.

An exploratory case study, as defined by Mills et al. [22], is particularly suitable for exploring areas where little prior research exists and where there are no established hypotheses, providing flexibility in the research design and data collection.

This exploratory case study, involving 50 university students in two groups, functioned as a workshop to investigate how software architecture students perceive, articulate, and organise natural questions and how their management should be conducted in systems. It encourages students to brainstorm and gather functional and non-functional requirements through a guided approach.

The main activities carried out during this workshop included:

- Acting as software architects, the students documented questions that emerged while designing an architecture.
- The students grouped related questions using sticky notes and visualised connections through spiderweb diagrams.
- Peer and instructor feedback was key to improving clarity and usability when presenting their findings.

This collaborative effort resulted in a categorisation of questions into six different categories: Users, AI, Categories, Tools/Functions, Answers, and Others. With this categorisation, each question would fit a broader category, which aids in identifying relationships and connections between similar or related questions. Each category captured distinct concerns and topics, which proved crucial in understanding the scope of the natural questions raised during the design process.

- **Users:** This category focused on understanding the users' needs and expectations and the different roles of users within the system. Questions in this group consisted of interactions with users, role definition and management, their interactions with the system and how the system could be designed to meet varying user requirements.
- **AI:** Questions within this category revolved around the potential integration of artificial intelligence tools into the system. Students explored and questioned how AI could support decision-making and task automation. This category also covered data privacy concerns and ethical considerations, focusing on the system's capabilities to connect the user to the AI productively.
- **Categories:** This category relates to the inquiries raised regarding the classification and categorisation of different questions and architectural concerns. The management and structure of questions were questioned, and how they should be presented was debated. Questions relating to the selected approach for the system being more in-depth and structured, or something to be defined by the user. This category helped identify key focus areas, such as scalability, maintainability, and categorisation.
- **Tools/Functions:** In this category, questions related to the tools, functions and technologies required to support the architectural design were gathered. The primary focus was ensuring that proper tools and functions would enable architects to capture, visualise and interact with their design decisions effectively.
- **Answers:** Questions within the answer category visualise how the system would respond to user queries and design questions. It also addressed how the system would act with unanswered questions and proposed adding a question backlog system. These questions addressed how architectural decisions should be validated and how their feedback would be provided. This category also emphatically demonstrated the importance of transparency and traceability in decision-making.
- **Others:** This category, Others, as the name suggests, includes miscellaneous questions that didn't particularly fit neatly into the previous categories while being extraordinarily relevant and vital to the architectural design. This category served as a catch-all for questions that required further exploration.

Figure 3.1 illustrates the results of the collaborative effort to understand, organise, and visually represent the relationships between natural questions raised during the architecture design activity. These spiderweb diagrams demonstrate how students grouped related questions into categories explained previously.

Several crucial questions were raised during the workshop, and through these discussions, newfound knowledge emerged, further emphasising the significance of questions in shaping software architecture decisions. The structured approach to categorising the questions clarified how

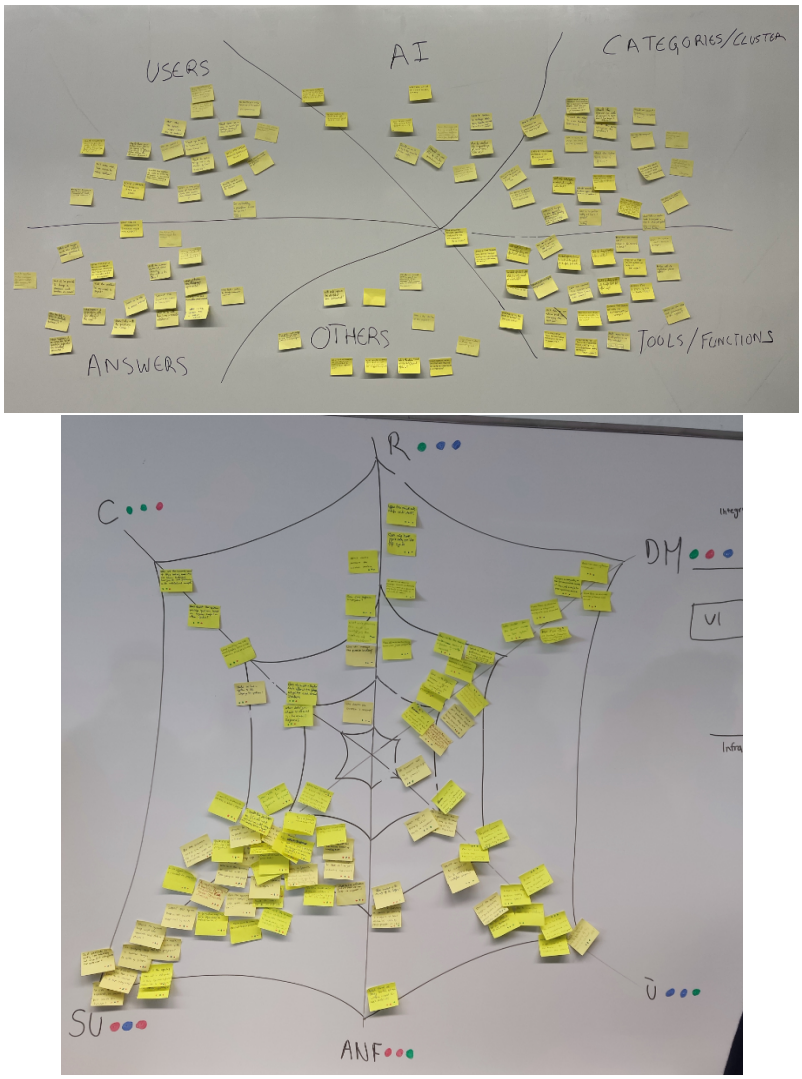


Figure 3.1: Resulting Spiderwebs

questions can be related to each other and highlighted the gaps and needs in the current question management systems. This case study stressed the necessity of a systematic, organised approach to capturing and tracking architectural questions to support informed decision-making throughout the questions' lifecycle.

3.3 Activity Diagram and Roles

The following activity diagram is a more thorough representation of the lifecycle of questions, compared to the flow chart diagram depicted earlier (2.1), focusing on the roles, phases and decision points inherent to this process. This diagram portrays the dynamic nature of the question's lifecycle, with several iterations and roles interconnected from its inception with its owner to the archival stage when deemed resolved.

The diagram highlights key phases such as question inception, prioritisation, collaborative discussion, decision-making, and archival, emphasising the iterative feedback loops between these stages.

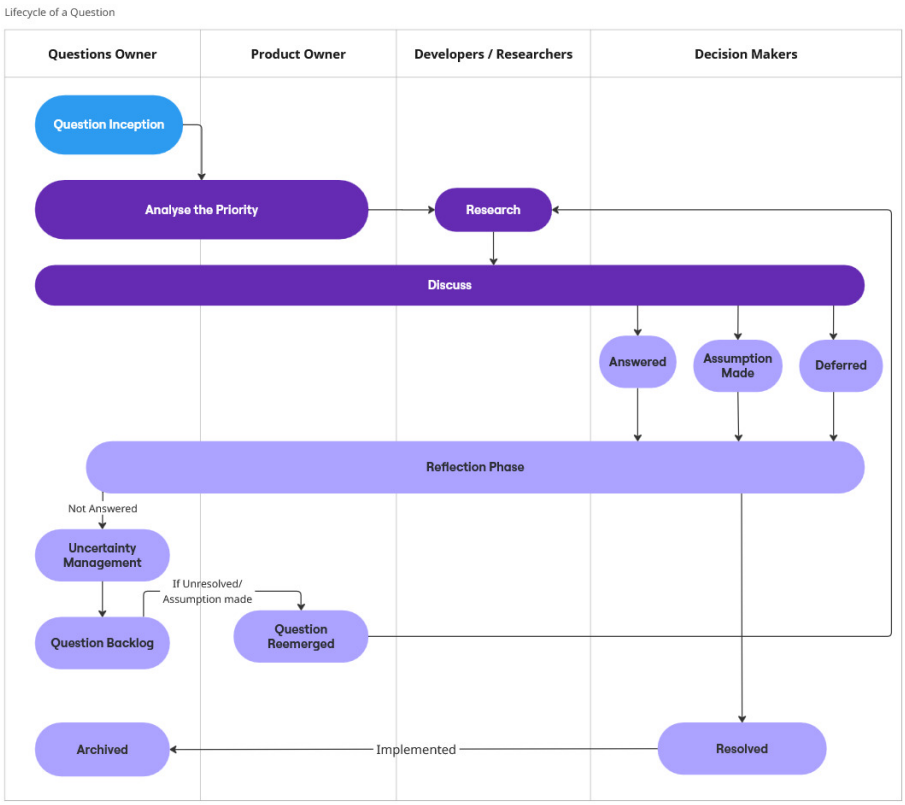


Figure 3.2: Lifecycle of a Question

The phases highlighted in Figure 3.2 consider the ongoing discussions for each question, often evolving multiple roles and the three possible outcomes of resolving a question. A question may be answered and resolved, it may lack a concrete and final answer at an early stage, but have an answer assumed to keep the system design and development moving forward, or it may remain unanswered at a given stage due to uncertainties or constraints.

Each role contributes distinct responsibilities to the lifecycle, as outlined below.

- **Questions Owner** Formulates the question and introduces it into the workflow, discussing it with other members to ensure it is clearly defined while also managing the question backlog and archival process.
- **Product Owner** Estimates the priority of the question in collaboration with the question owner, aligning it with project objectives, being a part of the discussion and reflection phase. When a question re-emerges, the Product Owner plays a key role in restarting the process. This role conforms with the Product Owner from Scrum [37].
- **Developers/Researchers** Conduct in-depth analyses and delve deeper to provide insights and solutions for discussion with other members.

- **Decision Makers** Evaluate all parameters and determine whether the question is resolved, deferred, or answered through assumptions.

Identifying these specific roles represents a nuanced procedure for comprehending architectural question management. While drawing from existing software systems management literature, these roles emerged through a systematic analytical process of deconstructing software development and architectural decision-making practices. This framework recognises that architectural questions involve complex interactions that traditional linear models cannot capture. Understanding these roles and dynamics is vital for resolving questions efficiently and preserving the rationale behind key design decisions for future development cycles.

It is important to note that while the proposed lifecycle presents a typical path of question management, it is not prescriptive. In practice, question resolution may occasionally deviate from this model, with potential different role interactions or unconventional routes that bypass standard communication channels. For example, in fast-paced agile environments, a developer may directly discuss with a stakeholder to determine a solution for a technical uncertainty. The framework recognises this flexibility, acknowledging that real-world software development processes often require adaptive communication strategies.

Through several iterations and with validation and expert feedback from both the survey and the workshop, this activity diagram to represent the lifecycle of questions was refined to reach a state where it is thought to be an accurate standard representation of how architectural questions are dealt with in any system.

3.4 Requirements

A structured set of requirements has been established to build a system that meets its intended objectives and effectively addresses the challenges identified in the earlier phases. These requirements were elicited and refined through relevant literature, the exploratory case study, 3.2, expert feedback gathered from the survey with software architecture experts, 3.5, and insights gained through a preliminary analysis of existing tools and practices.

The main problems encountered with current solutions were a lack of structured question tracking, fragmented documentation procedures and limited context preservation. Natural questions are qualitatively different from action items, requiring different ways to be managed and tracked: action items are tasks to complete something, usually known and natural questions are unknown and, therefore, often involve risks. Even if action items are to find something out, they are just one type of action item. They need special handling because of the risk, the inherent unknowns, and the specialised lifecycle.

Based on these insights, the specified requirements are a foundation for the system's design and implementation, guiding its functionality, behaviour, and performance characteristics. They are organised into two main categories:

- **Functional Requirements** — detailing the specific capabilities the system must support to fulfil its core purpose.
- **Non-functional Requirements** — specifying the quality attributes that must be present in the system, such as usability, performance, availability, and security.

The following subsections present each requirement in a structured format, enabling traceability and supporting the subsequent design and evaluation phases.

3.4.1 Functional Requirements

The functional requirements (FRs) define the core features and behaviours that the system must support. These requirements were organised into four categories and prioritised to ensure the full capacity of the intended tools. These categories include **Questions and Knowledge Retrieval**, **Classification, Prioritisation and Tracking**, **AI-Driven Insights** and **Collaboration and Security**. The following table, Table 3.1, provides a structured overview of these requirements.

FR ID	Category	Description
FR1	**Questions and Knowledge Retrieval**	The system must allow users to register questions via text and voice input to ensure that knowledge is captured in various formats, ensuring ease of use.
FR2	**Questions and Knowledge Retrieval**	The system should recognise images and diagrams, extracting relevant insights for question analysis and discussion.
FR3	**Questions and Knowledge Retrieval**	The system should transcribe audio recordings into structured text, automating transcription and filtering out irrelevant content to improve efficiency.
FR4	**Questions and Knowledge Retrieval**	The tool must provide searching, allowing for retrieving past questions through keywords, priority and issue.
FR5	**Classification, Prioritisation and Tracking**	The system must classify and store questions based on predefined categories and priorities to facilitate retrieval.
FR6	**Classification, Prioritisation and Tracking**	The system must maintain a question backlog consisting of active, resolved, assumed and archived questions.
FR7	**Classification, Prioritisation and Tracking**	The system must track decision-making processes, associating questions with choices and contributors.
FR8	**Classification, Prioritisation and Tracking**	The system must allow Product owners to prioritise questions based on urgency and impact to improve decision-making and resource allocation.
FR9	**AI-driven Insights**	The system must detect and highlight similar or ambiguous questions to reduce redundancy.
FR10	**AI-driven Insights**	The system must provide AI-generated recommendations for addressing architectural concerns and common pitfalls.
FR11	**Collaboration and Security**	The system should support multi-platform access to ensure availability.
FR12	**Collaboration and Security**	The system must notify users when a question is updated, commented or resolved to enhance collaboration.
FR13	**Collaboration and Security**	The system must implement role-based access control to restrict information visibility based on user roles.

Table 3.1: Functional Requirements

3.4.2 Non-Functional Requirements

To ensure high security, performance and usability, in addition to functional requirements, the system must meet these non-functional requirements (NFRs). These requirements address essential aspects such as how usable the system must be for any role, accessibility and security, which are critical to the system's adoption and long-term effectiveness. They set expectations for the system's behaviour while not defining specific features, as functional requirements do. Table 3.2 presents a structured summary of these non-functional requirements.

NFR ID	Description
NFR1	The system must quickly process user queries and responses to ensure real-time usability.
NFR2	The user interface must be intuitive for any role, requiring a small number of interactions for each action, regardless of the role.
NFR3	The system must be accessible anytime and anywhere, supporting the users and allowing spur-of-the-moment questions to be raised and addressed.
NFR4	All questions, decisions, and interactions must be secure, ensuring data integrity.
NFR5	The system should implement multi-level security, granting access based on user roles.

Table 3.2: Non-functional Requirements

3.5 Survey

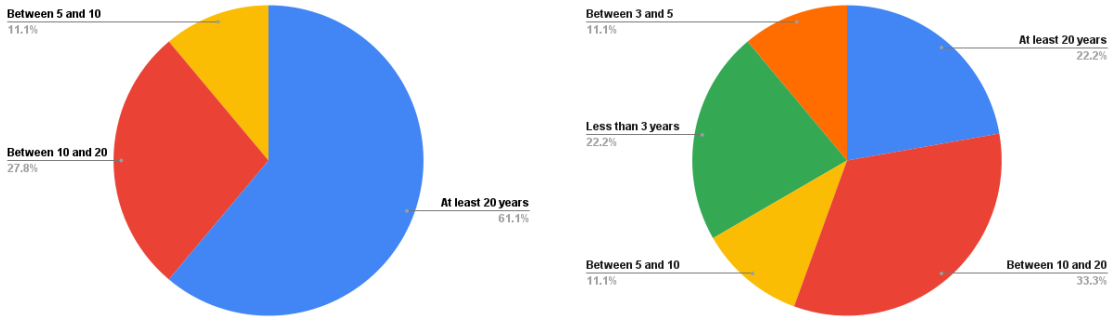
A crucial step to validate and identify additional gaps in this thesis is to gather expert feedback from experienced software architects and developers. The previous work and exploratory case study highlighted some key insights, heightening the understanding of the improvements and requirements necessary to create a relevant solution for managing natural questions. These insights suggest that AI tools and AI-based classification and prioritisation are key differentiators from existing tools. Multi-platform support that ensures accessibility is crucial for the adoption across teams and members. The system must act and intervene in discussions without hindering them and the natural processes that come with human interactions.

While the insights gathered from the workshop with students were vital, this next phase, a survey with expert entities, is paramount to validating the project and understanding its feasibility and industry relevance. Focusing on this task, a survey targeting professionals in the field of software architecture was conducted, aspiring to validate and refine our requirements and approach with expert insights. This survey was essential to collect quantitative and qualitative data on the lifecycle of architectural questions, the disadvantages associated with their management and the conceivable benefits of a structured tool for recurring tracking.

3.5.1 Demography

As stated before, this survey was geared toward experienced software architects with varying levels of experience and involvement in projects of various scales. This survey comprised 18 responses, with 16 respondents having at least 10 years of experience in software development and 11 having more than 20 years of experience. The experience as software architects yielded different results, with only six respondents having less than five years of experience, and 10 respondents having at least 10 years of experience in this role. As for project size involvement, participants worked on projects of various sizes, ranging from small-scale applications to large enterprise systems. Medium-sized projects rendered the majority of answers, followed by small-scale projects.

Figures 3.3a, 3.3b, and 3.4 present an overview of the respondents' profiles across these three dimensions discussed.



(a) Software Development Experience (b) Experience as a Software Architect

Figure 3.3: Professional experience of participants

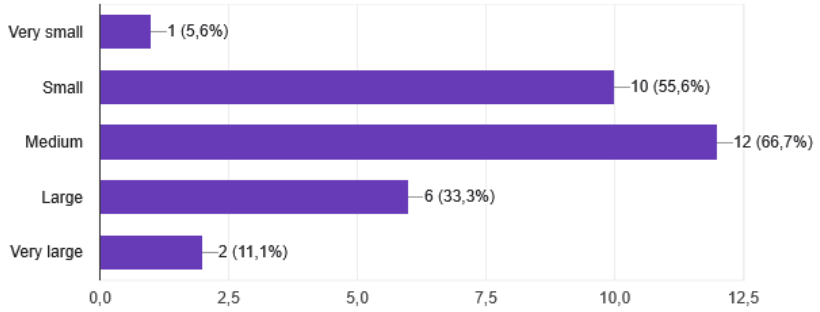


Figure 3.4: Typical project size experienced by participants

3.5.2 Difficulties when Managing Architectural Questions

After establishing the participants' demographics, they were asked if they had encountered challenges when managing architectural questions systematically, resulting in 16 out of 18 participants voting **yes**. Software Architects were then invited to identify the most significant challenges when

managing architectural questions. The results were distributed across all options, leading to the top three challenges reported:

1. Fragmented documentation practices suggest that existing documentation methods are inconsistent and lead to scattered information across different tools, documents and repositories. This fragmentation is a significant challenge when tracking and referencing architectural questions, discussions and decisions.

2. Difficulty retrieving past decisions implies that noteworthy architectural knowledge is often hard to search for due to a lack of proper management of questions that would render it available at all times.

3. Knowledge loss over time emphasises that architectural insights and reasoning behind decisions tend to disappear when not adequately recorded, especially in long-term projects or projects with rotating roles between members.

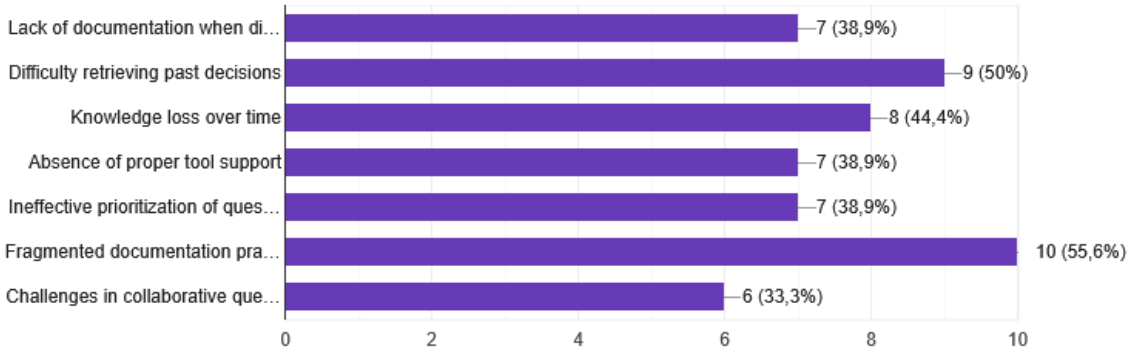


Figure 3.5: Most prominent challenges identified.

These insights support the pressing need for a structured approach to managing architectural questions, ensuring that key discussions and decisions remain accessible, well-documented, and preserved throughout the software development lifecycle.

3.5.3 Architectural Question Lifecycle Roles

Participants were tasked with evaluating the proposed roles in the architectural questions lifecycle presented as an earlier version of the activity diagram displayed previously. The responses were almost all agreeing in some capacity with the roles, with 10 agreeing and seven partially agreeing, with only one person not agreeing.

The architects were then asked to elaborate further on roles that are missing from the lifecycle and are deemed fitting. This question raised key insights, indicating a general alignment with the role structure while providing suggestions and refinements for role allocation. Some suggested changes for the role of product owner, questioning their participation in key areas and their responsibility. We can infer that while the roles mainly were agreed upon at a larger scale, there were

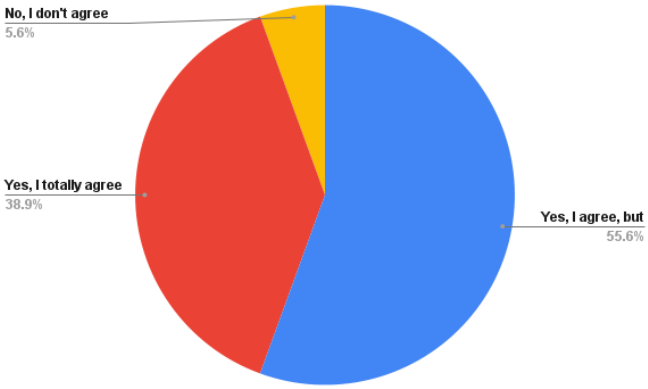


Figure 3.6: Activity Diagram Role Agreement

tweaks needed that would aid in refining and further completing the activity diagram, leading to the latest version, 3.2. The 12 suggestions for new roles highlighted how different every architect envisions the process we are trying to capture, and how hard it is to unite every vision of such a dynamic nature.

3.5.4 Architectural Question Lifecycle Activities

Relating to the same activity diagram, the participants were then asked whether the activities displayed in the diagram were agreed upon. This question resulted in the majority (17) finding this helpful classification and mostly accurate, with nine agreeing completely with what was displayed.

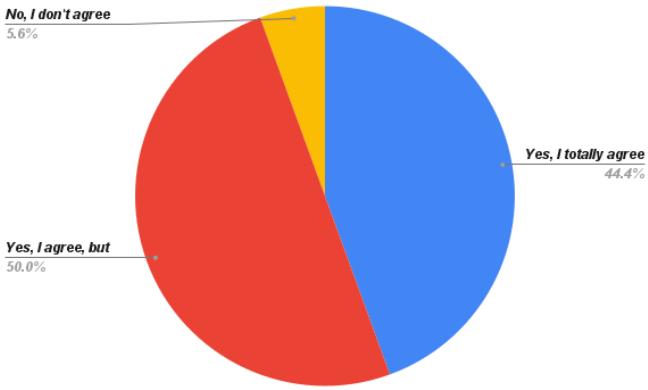


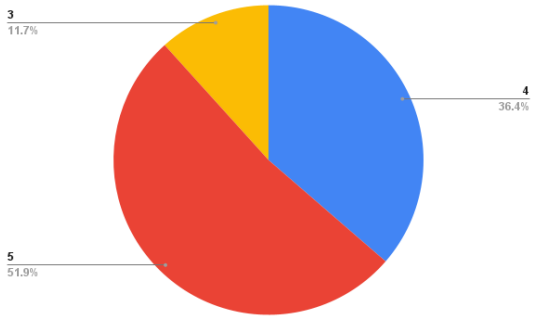
Figure 3.7: Activity Diagram Activities Agreement

Similar to the questions relating to the roles, the activities were deemed effective in capturing how questions evolve in practice, acknowledging the dynamic nature of questions. The diagram resonated well among architects, given its clarity in mapping the status of questions, and generated several suggestions from the participants. Among these suggestions were positive feedback and proposals for adding impact assessment activities when a question re-emerges and for a validation stage, which was implemented in the most recent version of the diagram. These findings were

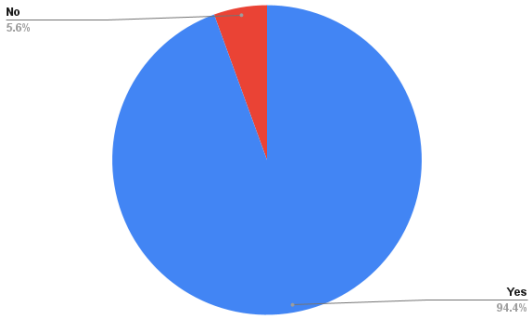
crucial to validate our classification and how well it was received, and they also added a new layer of depth to our work and diagram. All this feedback allowed for a more nuanced view on how questions evolve over time and ensures better tracking in the lifecycle.

3.5.5 The Importance of Recording Architectural Decisions

The following questions are essential to evaluate how valuable a structured tool for managing natural questions in software architecture would be. Firstly, the experts were asked to give a grade from one to five on the importance of systematically tracking architectural questions. Every answer was at least a 3, with eight people rating it a 5 out of 5, emphasising the agreement on how important the tracking of architectural questions is. After understanding how vital this tracking is, we inquired with the experts on whether they viewed the existence of a structured tool for natural question management in software architecture as necessary, which was answered yes by every participant, barring one (17).



(a) Necessity to Track Architectural Questions



(b) Demand for a Structured Tool

3.5.6 Critical Features Ranking

After establishing the perceived need for a structured tool to preserve and manage architectural questions and the knowledge inherent in them, the participants were tasked to rank the most needed and critical features for a solution to thrive, ultimately guiding the prioritisation of functionalities in the design of a practical and user-centred tool. A scoring system based on the positions ranked each feature in order of priority, and the most requested features were identified.

The five most requested features were as follows:

Question backlog management to track answered and unresolved ones. Ensuring that all architectural questions are systematically documented, monitored and reevaluated when necessary.

Architecture diagram recognition for visual knowledge extraction. Facilitating the interpretation and extraction of insights from architectural diagrams provides a more integrated method for managing architectural knowledge in the environment.

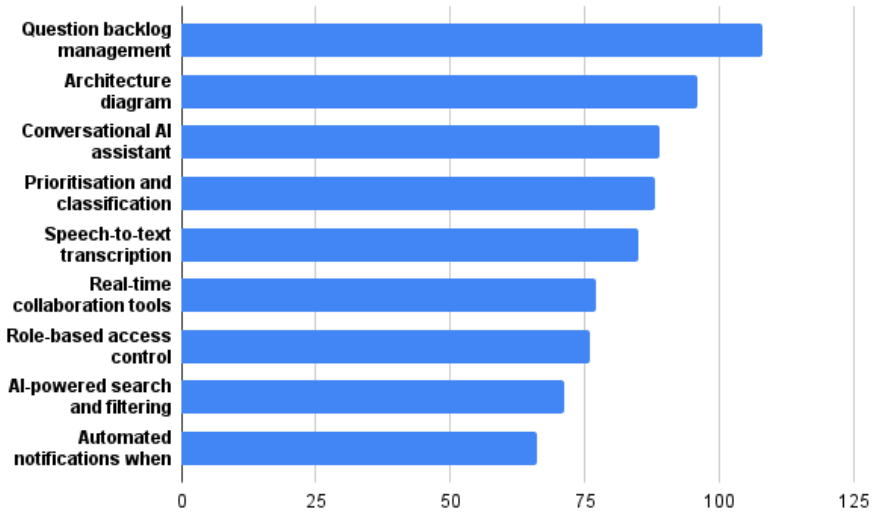


Figure 3.9: Ranking of Necessary Features

Conversational AI assistant to answer questions in natural language. Enabling swift and intuitive access to documented architectural knowledge via an AI-powered assistant capable of interpreting and answering natural language questions.

Prioritisation and classification system to identify urgent and high-impact questions. Develop mechanisms to evaluate and rank questions based on urgency, relevance, and potential architectural impact, supporting educated and timely decision-making.

Speech-to-text transcription for meetings and audio logs. Converting spoken discussions automatically into text to capture architectural questions, decisions, and rationale during real-time collaboration.

This question validates and displays the requirements and features that expert architects prioritise and look forward to when managing architectural questions, discussions and decisions. These requirements, elicited through literature review and discussions and from the exploratory case study, were now prioritised and allowed for the understanding of what is necessary for this system to flourish.

The survey was a critical step in building the best version of the system to maintain architectural knowledge. It facilitated meaningful discussions and provided insight into how important it is in software architecture, the essential features needed for its success and how different each expert views the dynamic nature of architectural questions, allowing for its enhancement, which led to its most recent version.

3.6 Problem Revisited

The root of the problem behind this thesis revolved around the observed gap in how questions, particularly natural questions, are captured, tracked and maintained across a project's lifecycle. These questions are pivotal in shaping architectural direction by clarifying ideas, identifying hidden assumptions and fostering critical discussion. However, their dynamic and scattered nature hampers their timely management and risks the loss of architectural rationale and design traceability [17].

Following the literature review, exploratory case study and survey results with experts in the area of software architecture, this assumption has been confirmed and refined. The literature review highlighted a significant gap in existing solutions for the management of architecture knowledge while also confirming its well-established importance throughout a project's lifecycle. Natural questions emerge as critical vehicles for capturing and conveying this architecture knowledge, establishing the importance of preserving them and their inherent wisdom.

The exploratory case study revealed that when tasked with a design scenario, software architecture students generated a wide array of important questions across several dimensions in the software architecture space. However, the main takeaway from this case study was that, without dedicated support, these questions were recorded inconsistently, scattered and grouped subjectively, being at risk of being overlooked and forgotten with time.

Likewise, the survey alluded to a consistent demand for a tool that would preserve questions, support their classification and prioritisation, and take advantage of AI capabilities for diagram recognition and speech-to-text transcription. Experts emphasised the importance of such a tool, which would fill a gap in a known but underexplored problem. The lack of such structured support was especially damaging in fast-moving or distributed teams, where asynchronous collaboration increases the likelihood of fragmentation.

This research further established how existing methods, from sticky notes to unspecialised tools, fail to give questions their true significance. This reinforces the need for a purpose-built solution that treats questions as the dynamic and evolving artefacts they are in the architecture process.

Following these findings, the original problem is validated and sharpened further. The absence of structured support for architectural question management is not just a minor oversight but a damaging and systematic weakness that hampers architectural knowledge sharing and retention and provokes a loss in long-term design integrity. The problem clarification and insights gathered in this chapter informed the next stage of this work, developing a tool designed to capture the lifecycle of natural questions and preserve them accordingly.

Chapter 4

System Architecture and Prototyping

Building on the evolved understanding from the previous chapter, this chapter translates those insights into a concrete system design. Aiming to address the clarified problem, this solution seeks to establish natural questions as persistent, structured, and context-aware artefacts within the software architecture process. An AI-enabled tool to support the capture, classification, discussion and long-term traceability is paramount.

This chapter presents the system architecture, prototyping process, and execution of the proposed solution, which is designed to support the capture, management, and preservation of natural questions in software architecture. This solution is grounded on the critical challenges identified in earlier chapters by leveraging a platform that integrates AI-assistance, structured question tracking, semantic understanding and focuses on the collaborative process that natural questions incite.

The first step was to outline the architecture, highlighting its modular structure and key components, integrating AI-driven classification, semantic search, and decision tracking into a cohesive, role-abiding platform. The initial low-fidelity prototype is a foundation for an environment capable of processing questions from inception to archival. The fully functional implementation was developed through several technologies such as Python, Streamlit, FastAPI and Google's AI services.

Finally, we present the case study that illustrates the application of the system in a real-world context, using data collected from a Master's degree course on Software Architecture. With more than 400 questions collected, the evaluation and implementation phase was built upon these questions and the knowledge inherent in them.

4.1 Architecture

The architectural design of our proposed system represents a methodical approach to addressing the intricate challenges of question management in software architecture. The architecture integrates AI-enabled processing with thorough question lifecycle management and secure, role-based access control by creating a service-oriented approach.

The system's core components include a central Question Management system that coordinates information flow, a decision-tracking mechanism to preserve discussion logs, a multi-level role system ensuring access control and AI services that provide heightened research, retrieval and categorisation. This architectural approach bridges the gap on previous limitations in question management by enabling structured question tracking, supporting multi-modal input processing, and facilitating comprehensive knowledge preservation.

This architecture represents an adaptive environment that transforms how architectural questions are captured, analysed, and managed. Ultimately, it fosters a more structured, transparent, and intelligent approach to handling architectural questions, improving collaboration and decision-making throughout the software design process.

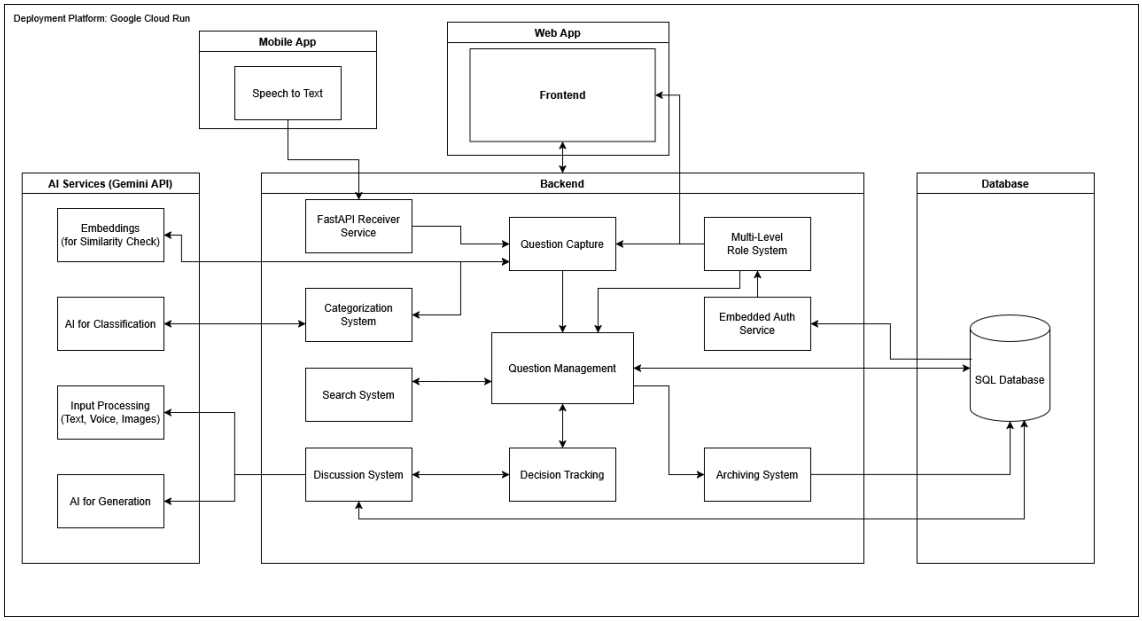


Figure 4.1: Architecture

4.2 Low and High Fidelity Prototyping

Following the Design Science method referenced in section 3.1, we designed and iteratively refined prototypes to build upon the tool we envisioned to preserve and manage natural questions. In this phase, we focused on defining core functionalities, developing mock-ups and aligning the features with the identified requirements.

Two low-fidelity prototypes were initially created using digital sketching techniques to conceptualise the overall tool. The first iteration represented the tool in a chat-centric way, with filtering and a backlog of questions. This design aimed to encourage structured discussions and historical tracking. However, this approach was limited upon further refinement, particularly in capturing the high volume and scattered nature of the architectural questions. The second design resembled a spatial visualisation, further explored in a higher fidelity prototype developed in Figma.

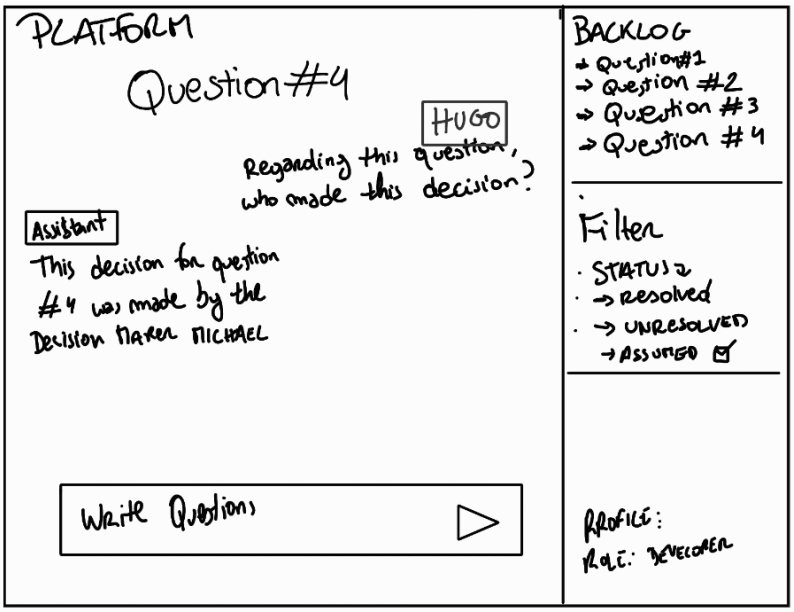
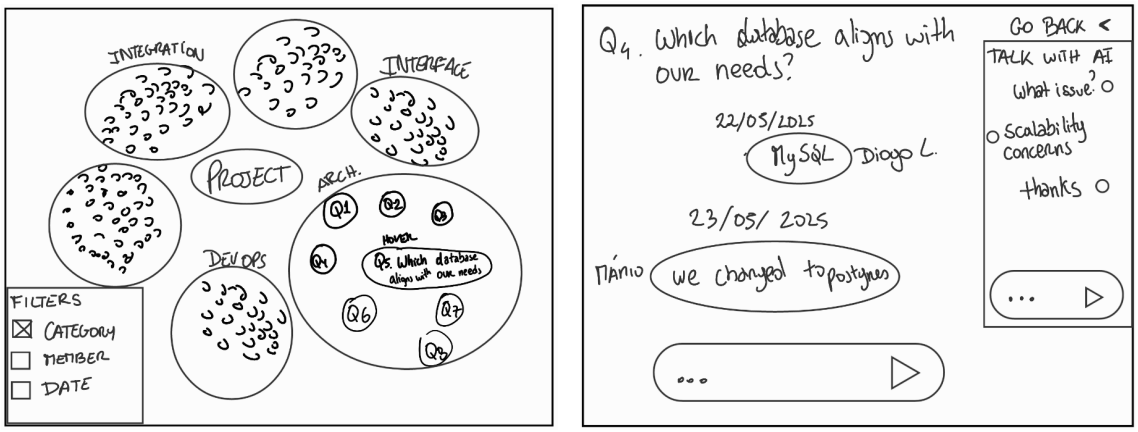


Figure 4.2: First iteration of a Low Fidelity Prototype



(a) Spatial Visualisation of Questions

(b) Chat interface for Low fidelity prototype

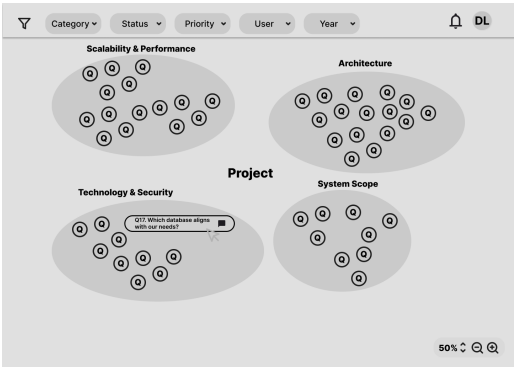
Figure 4.3: Second Iteration of a Low Fidelity Prototype

Building upon the low-fidelity prototypes, we explored a categorisation-based interface that enables users to visualise questions spatially within distinct clusters. This new approach enhances retrieval by allowing filtering based on specific parameters. It exposes the relationships between different questions, which is highly beneficial for decision traceability, knowledge structuring and identification of connected questions.

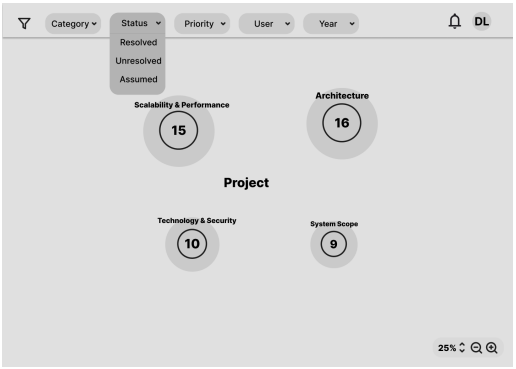
Building on this foundation, we developed a high-fidelity prototype using Figma, using interactive elements to capture the refined usability and representation of categorised questions. Insights from the literature review, survey and workshop influenced key design decisions.

This exploratory iterative process in the low-fidelity phase laid the groundwork for a more robust and user-centred solution. Whilst the insights from the simple chat-centric model are still

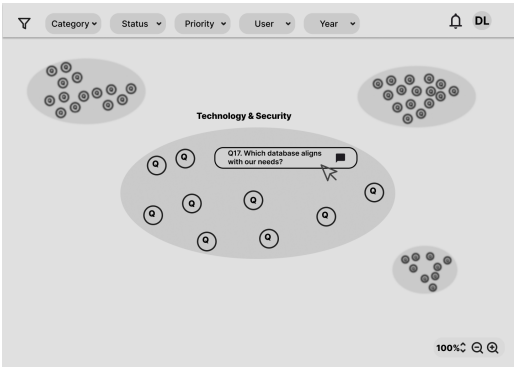
present, the categorisation-based interface clarified the priorities and essential features needed to support the lifecycle of architectural questions effectively. This progression, grounded on design science iterative principles, consolidated these insights and nurtured the transition to functional implementation, which is detailed in the next section.



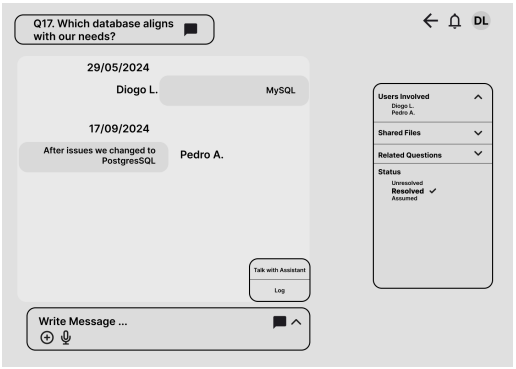
(a) Overview of the categorised question clusters



(b) Zoomed out visualisation of the categories



(c) Detailed view of questions inside a category



(d) Chat with members and AI integration

Figure 4.4: Screens from the high-fidelity prototype built in Figma

4.3 Functional Prototype

With the requirements refined by the several previous stages explained in detail in Chapter 3 (literature analysis, workshops and interactive feedback), the goal of the full implementation of this tool is to deliver a functional prototype that encapsulates the proposed lifecycle for managing architectural questions, 3.2, and addresses the key limitations identified in existing tools as summarised in table 2.3.

This prototype integrates core backend features with a user-focused frontend interface, focusing on effectively capturing, classifying, prioritising, discussing, and managing questions. The following subsections describe the technologies used, the frontend interface, the backend logic, AI integrations and the different pages created to support the end-to-end lifecycle of architectural questions.

The complete source code for the prototype can be found in Appendix A.

4.3.1 Frontend

The frontend of our solution was implemented using Streamlit 4.3.1, facilitating the rapid development of an interactive, web-based interface entirely in Python and allowing for tighter integration with backend services. The user interface was designed with clarity and usability, considering a research prototype context, supporting every user regardless of their technological background.

The main tasks included capturing new questions, enabling search and filtering, visualising the queries and the inherent metadata in a board-like interface, and supporting discussions with AI and coworkers. This layout facilitated a structured interaction for the users to navigate seamlessly across multiple questions, empowering the user with semantic search, backlog management and AI-assisted dialogue.

Streamlit

Streamlit [35] consists of an open-source Python framework for building interactive web applications. It is mainly used for data-driven and machine learning projects since it allows developers to create user interfaces using Python scripts, reducing the workload of developing a complex frontend.

Streamlit was one of the options that stood out in building this tool, and it was selected as the primary interface framework due to its rapid prototyping capabilities and seamless integration with Python-based backends. It aligned with the real-time interactive nature of this tool. It excelled due to the fast integration with various Google Generative AI services, enabling cohesive and efficient communication between the user interface and AI-powered backend components.

4.3.2 Backend

The backend component of our tool is responsible for managing the core logic, data storage, data persistence and integration with external services, such as Google AI services. It functions as the orchestrating layer that links the user interface with our developed functionalities, namely the question classification, semantic searching, discussion tracking and many more. Implemented entirely in Python, it is designed with modularity and maintainability, facilitating rapid interaction and reuse, which is essential in experimental, research-oriented prototypes.

Python Backend

As stated before, the backend is implemented in Python; this language was chosen for its versatility and, more importantly, due to the strong ecosystem of libraries for natural language processing and web integration. Google's Generative AI SDKs being compatible with Python was key to this decision, allowing the use of advanced language models and embedding APIs without additional issues. The organisation of the backend is standard, with the separation of the backend and the frontend. Inside each, it is separated into separate modules (e.g. ai.py, auth.py, ui_question.py), each dedicated to a single responsibility. This organisation sticks to the principles of separation

of concerns and allows easier testing, debugging and overall scaling. Python's ease of integration with SQL databases was also fundamental, combined with its rapid prototyping strengths, making it an ideal fit for this tool development, where experimentation and adaptability were crucial.

Google Generative AI API

Google's Generative AI APIs were fundamental to this tool. The backend integrated them to leverage several core features, such as automatic classification and prioritisation, semantic similarity detection, and natural language discussion. The model selected for prompt-based generation was the "gemini-1.5-flash-latest" due to its speed and competitive output quality. It successfully generated category labels, taxonomy tags, priority assignments and natural language responses in back-and-forth discussions on software architectural themes. For similarity detection, the model used was the "gemini-embedding-exp-03-07" to encode high-dimensional semantic embeddings of questions, which are then stored and compared with each new question, to identify potential duplicates or related entries using cosine similarity. The decision to use Google's models was based on their high performance, cost-efficiency, low latency and seamless integration via Python APIs, which reduced implementation complexity and ensured a natural integration with the overall system architecture.

Embeddings

As described previously, this tool generates and utilises vector embeddings of question texts to support duplicate detection. These high-dimensional vectors are stored in the SQL database using the embedding model to enable fast, repeated comparisons without overloading the system with redundant API calls. Semantic similarity is computed using cosine distance and has a defined similarity threshold that balances recall and precision in detecting near-duplicate or related questions. This threshold was used to detect similar and almost identical questions on related affairs, which the user was advised not to insert as new questions. Given a question that would fall into the 80th to 95th percentile, it would flag the user with a similar question warning and display the relevant matches, allowing for the manual assessment of the question to determine if it is adjacent to the ones shown or sufficiently different to warrant inclusion. Questions above the 95th percentile were automatically rejected, given how close they were to existing inquiries. This design ensures the automation process is efficient, allowing user input to detect questions that may fall into similar phrasing, with different intent or context. This embedding layer is crucial in maintaining the quality and coherence of the questions' backlog by encouraging reuse and detecting similar entries that could waste some resources.

Prompts

Given the several tasks highlighted in this tool that need AI intervention, prompts were crafted to guide the behaviour of the generative AI models. Having context-awareness and relevancy as the priority for the AI responses, each prompt was carefully designed to align with the research

goals and the software architecture domain. The prompt engineering process was iterative and informed by empirical observations during several internal prototype testing, understanding how specific each answer should be, keeping in mind each role defined in Section 3.3—such as the Question Owner, Product Owner, Developer, and Decision Maker. Given the different roles implemented, the prompts would vary accordingly, with a higher degree of technological knowledge for developers and a more surface-level technicity, but more suited advice for a role such as Product Owner. This structured interaction with the AI models is essential for shaping the quality and usability of suggestions and discussions for any question. The prompts that were developed are in Appendix B.

Database

The database employed in this tool was the lightweight SQLite to persist core application data. It handled the sets of questions, their associated metadata (such as owner, status, priority, tags), semantic embeddings for similarity comparison and discussion threads. SQLite proved to be the most pragmatic balance between simplicity, performance, and reliability, with the ease of upgrade if full-scale database management is necessary. Its local caching of embedding vectors was crucial for fast similarity comparisons and full-fledged experimentation. The schema was designed to be modular and extensible, given the potential transition to a more robust database solution if the system requirements evolve drastically. The main tables were questions and discussions, each representing the questions and their metadata, and the discussions represented the discussion thread with a foreign key to the question in which it was inserted.

4.3.3 Pages

The following subsection highlights several pages designed for the prototype to support a specific stage of the architectural question management lifecycle. The user interface was focused on being intuitive and interactive to provide the user with a less cognitively burdensome tool and more efficient for managing complex architectural questions. This philosophy is crucial, as the consistent use within the architectural design process can determine the tool's effectiveness. If the interface were too complex or cumbersome, it would defeat the primary objective of streamlining architectural question management and encouraging continuous management of natural questions.

Questions' Backlog Board

The home page of this tool is the Questions' Backlog Board and serves as the central interface for managing all architectural questions. These questions are organised into distinct columns based on their current status: "Open", "Assumption", "Resolved" and "Deferred". These statuses were previously defined as the core status in the lifecycle of a question.

This organised board facilitates the efficient review of all questions and allows for a comprehensive examination of their metadata through interactive, clickable cards. Each card, representing a different question, displays the author, category, taxonomy, priority, creation date and status.

Furthermore, a role-based access system allows authorised users to change the priority and status of a question, further aligning with the lifecycle of natural questions and responsibilities for each role, 3.3.

In anticipation of a substantial volume of questions within each project, a searching and filtering system was developed to overcome the limitations of traditional retrieval systems. This searching system allows for the conventional free-text search with character recognition and formulating structured queries using filters such as prio:High, owner:Diogo, tag:Design, status:Open, and category:Testing. These filters can also be combined with spaces and commas for "AND" and "OR" logic, respectively.

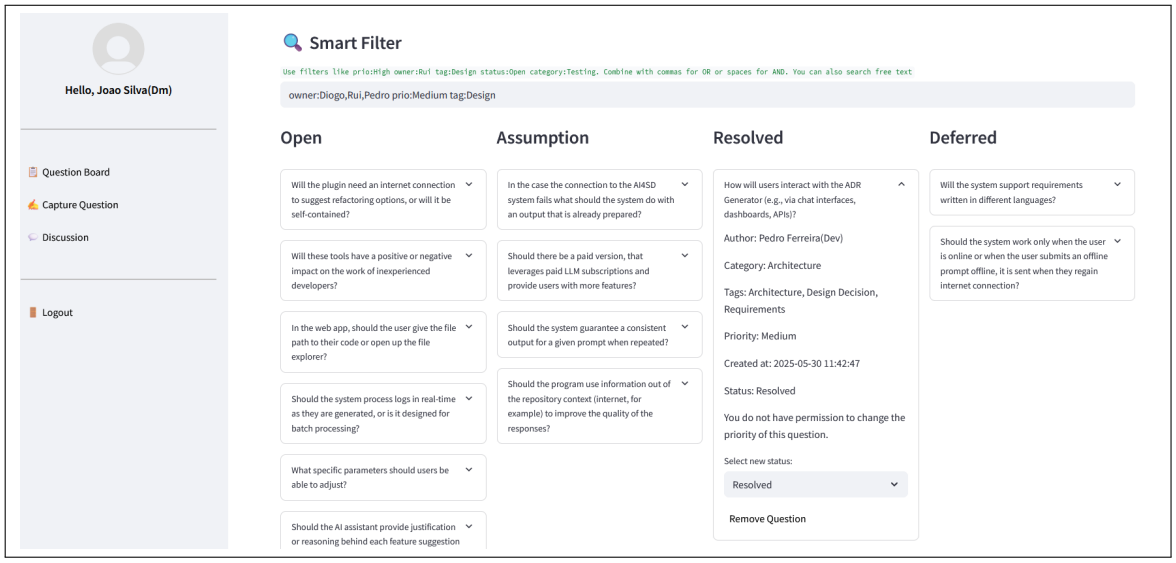


Figure 4.5: Questions' Backlog Board

Capture the Question

Capturing new questions is the first step in establishing a structured architectural decision-making process. This interface is designed to accommodate questions manually or through voice transcription (explained in detail in section 4.3.4), ensuring accessibility across devices and contexts. If the question is captured through the mobile app, a "Load Question from Mobile App" button is available, which, when clicked, pre-fills the input field with the transcribed content, facilitating the submission process.

Upon submission, the input is automatically analysed and compared with the entire backlog to find similar or equivalent questions. As explained in section 4.3.2, this result is obtained with an AI embedding model, which displays the results that exceed a similarity threshold. If it is similar, the user can cancel the question or submit it regardless, if the user perceives the question as necessary and sufficiently different from the ones displayed as identical, as seen in figure 4.6.

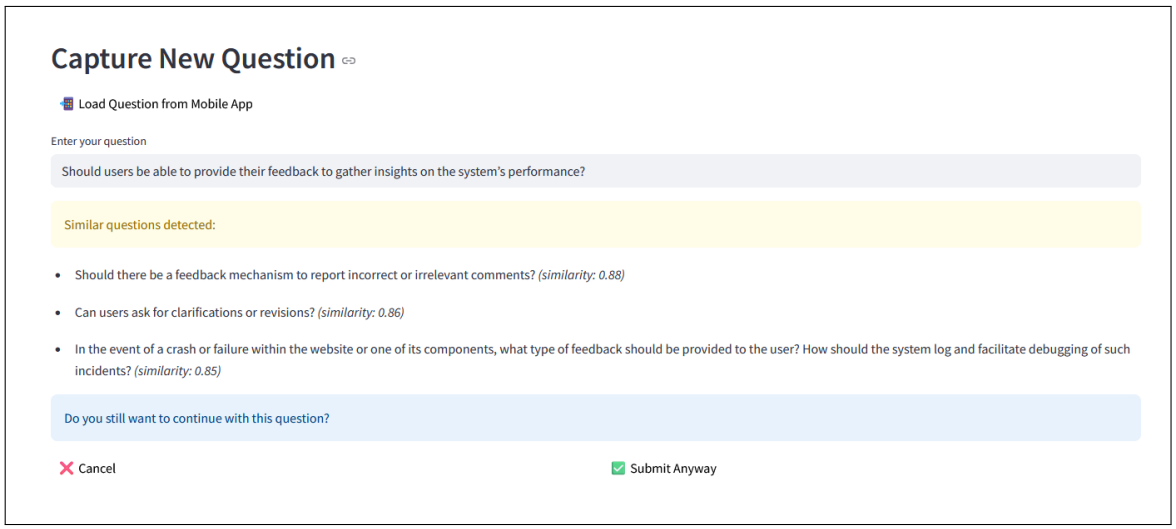


Figure 4.6: Similarity Check on New Question

After confirming the submission as a new question, the system initiates an AI-enhanced classification process that assigns relevant metadata on the question's context. The system detects the category, taxonomy and priority based on the question's content and the project's overall context. This is key to maintaining and retrieving questions throughout the project lifecycle. The category and priority of a question can be manually adjusted after submission, recognising that their relevance may vary depending on the user interacting with the question.

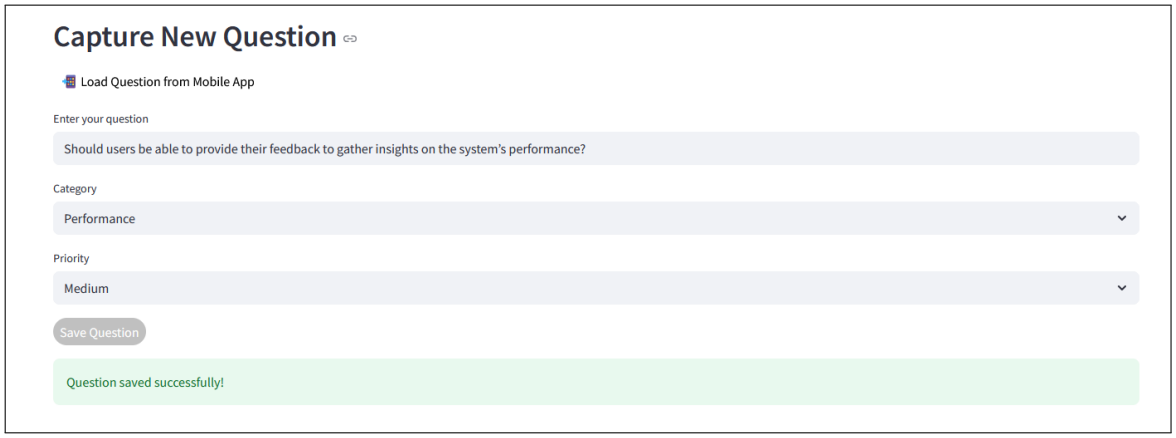


Figure 4.7: Capturing a New Question

Discuss the Question

The value of questions can not be measured only by the issues they raise, but also by the discussions they spark. In these questions, trade-offs, design rationale and hidden assumptions are often revealed and prove fundamental to the architectural decision-making.

Each question in the backlog has a dedicated, threaded discussion interface to create an environment suited for this discussion and reflection process. Users can initiate and contribute to

discussions by adding comments, insights and questions to be shared and stored throughout the question's evolution, fostering architectural reasoning and accountability for each decision.

To further support this process, an integrated role-based AI assistant can respond to prompts, offer rationale, synthesise advantages and drawbacks, summarise prior messages and anticipate possible outcomes for each decision. Each role has its dedicated AI assistant, tailored to mirror the responsibilities, priorities, and concerns associated with that role, ensuring more relevant and role-aware architectural discussions. These conversations are persistently stored in the database, maintaining crucial data such as the date and user for each message, allowing for the traceability and preservation of past rationale. In addition, the AI can process and interpret uploaded diagrams and images, providing contextual insights that are key to discussions.

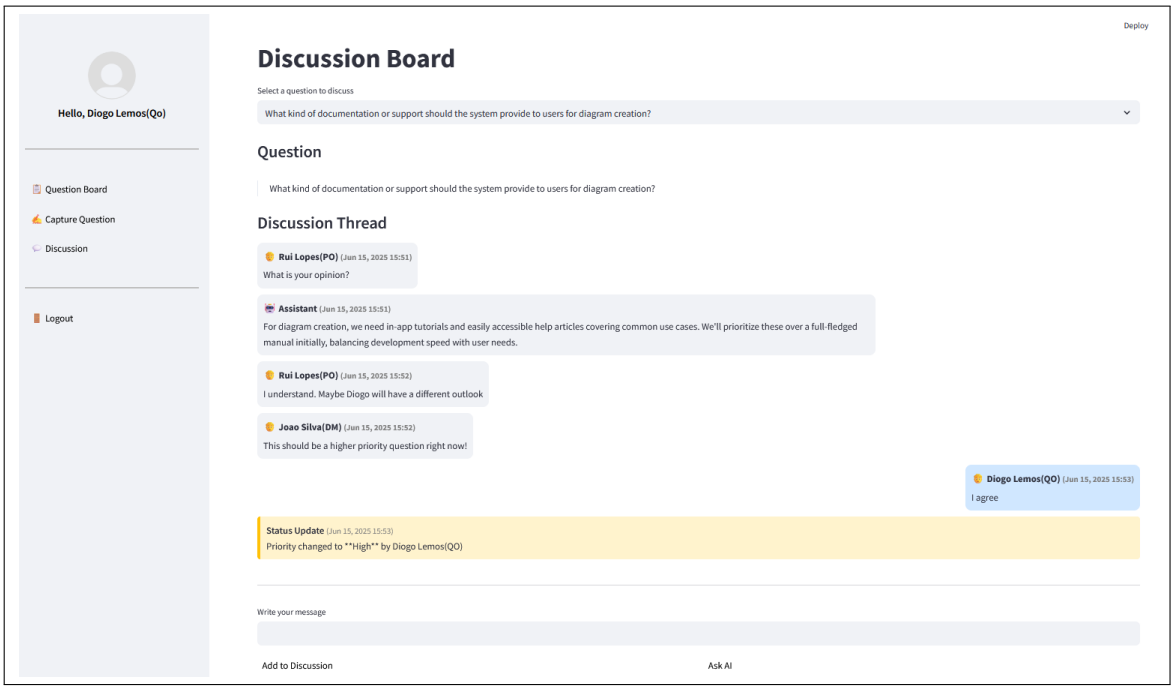


Figure 4.8: Discussing a Question

Authentication and Role-based Access

Given the importance of role-based functionalities within the tool, a basic authentication system was implemented using a lightweight YAML configuration file. Although intentionally simple, this system allows for a structured login mechanism that distinguishes between roles and accordingly governs access to specific features. Every user can submit new questions and contribute to ongoing discussions, which are role-aware when speaking with the AI assistant. Each role has its permissions given its importance at the different stages of the question's lifecycle.

- **Questions Owner** and **Product Owner** have the ability to change the priority of each question in either the discussion after new insights or on the main board cards. The AI assistant for the role of Question Owner focuses on architectural quality attributes, trade-offs, and

long-term implications. In contrast, the product owners' AI assistant concentrates on business value, stakeholder alignment, and clarity of requirements.

- **Developers** and **Researchers** are met with an AI assistant focused on supplying information on implementation details, trade-offs, and feasibility.
- **Decision Makers**, responsible for the decisions after all the parameters are evaluated, can modify the question status. Their AI assistant thoroughly evaluates the entire conversation while focusing on strategic direction, risk, and prioritisation.

Essentially, all submitted content is tied to the authenticated user's identity, following the functional and non-functional requirements and allowing for transparent traceability of authorship and accountability throughout the question's lifecycle.

4.3.4 App

Natural questions are the questions that naturally arise in one's mind throughout the software design process. These questions can emerge at unpredictable moments, during informal conversations, commutes or between meetings. To support this process of capturing and maintaining every question materialised in the several development phases of a project, we believe that it would be highly beneficial to support the primary tool with a lightweight phone app that enables quick capture and seamless transmission of questions to the central system.

This app 4.9, created using Flutter, allows users either to write questions or to dictate them using speech-to-text functionality. Once captured, questions can be transmitted via an API connection to the central platform, where they are integrated into the questions' backlog.

Flutter Voice Interface

The mobile interface is intuitive and minimalistic, featuring a voice recording button, a live text feed in which you can edit the text that has been recorded, an archive of saved questions and a send button. Users are prompted to click the microphone icon to start recording questions. While recording, the words are instantly transcribed using speech-to-text technology, and the resulting text is displayed for review or editing before saving or submission.

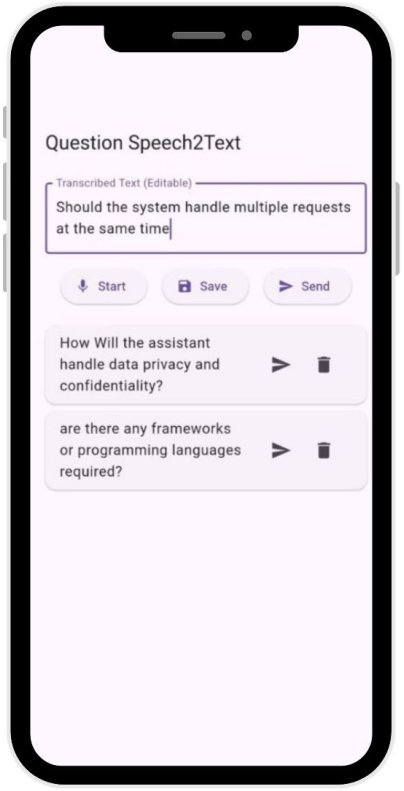


Figure 4.9: Mobile app interface for capturing voice questions

Transcription and API Connection

After recording and sending the question through the app, the input is encapsulated into a structured JSON payload and transmitted securely to a lightweight intermediary receiver server via a secure HTTP POST request. This occurs through an intermediary since Streamlit applications don't natively support persistent, stateless API endpoints for external ingestion. To address this limitation, a FastAPI service is a dedicated gateway for receiving and forwarding mobile submissions to the primary backend. This asynchronous interaction aims to maintain responsiveness and easily capture ideas anytime, anywhere.

The mobile app is a crucial part of the question capture workflow of the main platform. Providing a feature that the current tools do not offer ensures that the dynamic nature of questions can be recorded in various contexts.

4.3.5 Deployment

In this project, deployment was a crucial concern, requiring scalability and support for rapid iteration. Google Cloud Run was selected as the tool's hosting environment to meet these requirements.

Cloud Hosting

Google Cloud Run is a fully managed compute platform by Google that automatically scales stateless containers and executes them in response to HTTP requests. Suitable for event-driven systems, this platform was chosen as it abstracts away server management while also supporting containerised applications. Docker is used to containerise the system, ensuring portability and reproducibility across environments, simplifying deployment while preserving the scalability with minimal operational overhead.

Given Streamlit's limitations regarding API routes, two separate services were deployed to Cloud Run. The primary service hosted the web interface while a secondary FastAPI-based service acted as a dedicated HTTP endpoint receiver for external inputs, such as the speech-to-text mobile app.

The deployment model supported remote accessibility, iterative development and extensibility with minimal infrastructure complexity. Given the need for a Google Gemini API, a personal key was stored safely for the case study and early development to enable external users to utilise the AI functionalities fully.

4.4 Case Study

To ground the prototype in a realistic and relevant context, we used an academic project of the Large Scale Development course at FEUP as a case study. The AI4SD (AI for Software Development) is a project involving over 25 student teams developing AI-powered assistants targeting the different phases of the software development lifecycle, including architecture and design.

Given the project's setting, the AI4SD context is a rich and lifelike environment to gather real-world questions regarding design decisions, AI integration, and system structure. Over 650 questions were collected to populate the application's initial dataset with representative and authentic architectural inquiries.

Using this case study ensures that our research is theoretically and empirically anchored in a realistic software design scenario, aiming to increase its validity and practical relevance.

Chapter 5

Evaluation

The main goal of this thesis was to design and implement a tool to support the maintenance and preservation of knowledge acquired from the dynamic process of natural questions during software architecture design. We conducted a controlled experiment using a case study to evaluate the tool's effectiveness and practical utility. The objective of this experiment was to assess how competently the tool supports the capture, classification, and preservation of architectural questions and their rationale, compared to traditional methods.

In this chapter, we describe the empirical methods carried out in the evaluation, including the objectives, research questions, planning, the tasks performed, and the results obtained, followed by an extensive analysis and discussion.

5.1 Objectives

This evaluation assesses the proposed tool's effectiveness, usability, and practical relevance to capture, preserve and manage naturally emerging architectural questions. Through a controlled experiment, based on a developed case study, we defined three tasks to evaluate whether the tool enhances how participants capture, organise, and decide architectural questions that surface during software design. The developed case study, alluded to before in section 4.4, is used in this case due to the familiarity of the case study with the group of participants, given that the case study was initially developed by a group of university students to research architectural questions, making it particularly relevant and relatable to the context of this project and to the participants involved.

Ultimately, there is a need to understand how valuable the participants find the tool, its usability, its value in the landscape of software architecture, and its applicability in real-world software architecture tasks and workflows.

5.2 Research Questions

We formulated the following research questions to conduct the evaluation:

- **RQ1:** Does the tool enable users to capture more relevant and complete architectural questions compared to traditional methods?
- **RQ2:** Does the tool improve the organisation and classification of questions using taxonomy and tags?
- **RQ3:** Does the tool facilitate swift or more effective resolution of architectural questions?
- **RQ4:** Does the tool support the preservation and retrieval of architectural knowledge for future reference and decision continuity?
- **RQ5:** Do users find the tool usable and helpful in supporting architectural design activities?

5.3 Participants

For the tasks, the core participants are the following:

Student Group: Master-level software engineering students with prior exposure to software architecture. Participants were recruited from a university course that delved into architectural questions and were familiar with the case study, allowing for the completion of design tasks using the tool.

Experienced Software Architects: Practising software architects with significant professional experience in system design and architectural decision-making. Many of the participants in this group were also the experts contacted in the previous survey. Their involvement provided insights into the tool's effectiveness in real-world scenarios.

5.4 Tasks

- **Task 1 - Capture and Classify** Participants understood the scenario and captured relevant naturally emerging architectural questions through the tool. They interacted with the AI-assisted classification, which automatically suggests categories, priority and taxonomy. Participants could accept or adjust the classification and prioritisation as needed.
- **Task 2 - Searching and Managing the Board:** Participants were tasked to explore the four roles acting on the tool (Question Owner, Product Owner, Developer and Decision Maker) and their capabilities, through modifying the priority and status of questions. Participants are also tasked with searching using the smart filtering system with free text and tagging system (ex: prio:High owner:Rui).
- **Task 3 – Discuss and Resolve:** Participants selected one or more critical questions and used the tool's discussion feature to engage in a simulated collaborative discussion or explore AI-generated advice and commentary through questions. They documented their decisions and rationale as part of the resolution.

5.5 Procedures

The evaluation was conducted in a controlled setting, and participants completed the tasks proposed. At the start of the session, through a brief questionnaire, we collected the demographic information of each participant, including their experience with software architecture and the themes involved. Participants were then briefed on the purpose of the study and given a brief tutorial on how to use the question management tool. This included the available roles, authentication, and a very short overview of the system's capabilities without hampering the exploratory steps needed for the evaluation.

Participants were then guided through the three tasks described in Section 5.4, using the same case study to ensure comparability when analysing results, logging their actions, including time spent on each task, changes made and interactions with discussion or AI features.

Participants were encouraged to explore the tool through the lenses of the different roles representing the leading actors in a real-life system, and instructed to operate within their role's permissions.

After completing the tasks, participants completed a post-task questionnaire to evaluate their thoughts and perception of the tool's usability, usefulness and feature clarity.

5.6 Metrics

Following the controlled experience, evaluation metrics were defined to grasp the tool's usability, effectiveness and perceived value from the perspective of potential future users. To evaluate the study's outcomes, qualitative and quantitative metrics were defined, aligned with the experimental tasks stated previously, 5.4.

Firstly, we gathered some background information, including their profile(e.g., Student or Acting Software Architect), total software development experience, and experience as a software architect. The three major tasks(question capture and classification, board management, and AI-assisted discussion) required a self-reported time completion(in minutes). Participants responded to a series of questions using a 5-point Likert scale for these three tasks. These scales were used to measure aspects such as:

- Ease of capturing a question.
- Accuracy and usefulness of AI-generated classifications.
- Intuitiveness of the board interface.
- Effectiveness of search and filter functionalities.
- Clarity and usability of role-based permissions (e.g., Product Owner vs. Decision Maker).
- Helpfulness of AI assistant responses and support for reasoning.
- Overall intuitiveness of the tool.

Likert responses were measured on a 5-point agreement scale:

1	**2**	**3**	**4**	**5**
Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree

In addition to these responses, participants were invited to provide free-text comments after each task and at the end of the experience. These insights captured users' impressions of the tool's most valuable features, limitations encountered and suggestions for further improvement. The final question was whether participants would consider using such a tool in their professional settings.

5.7 Result Analysis

This section presents the findings from the user study conducted. The evaluation was divided into four main parts relating to the three tasks and a final reflection. As stated in the metrics, the results retrieved followed a Likert-scale and also open and closed-ended questions.

Demography

Following the definition of this evaluation, the participants tasked with answering this form primarily consist of university students and software professionals, including software engineers and experienced software architects. Most respondents had between 3 and 5 years of total software development experience, with a few participants having extensive experience (20+ years) specifically in software architecture.

Task 1 – Capture and Classify

Participants generally found it straightforward to capture questions using the tool, rating ease of use positively. Users completed this task quickly, with most participants reporting a time investment of just two to five minutes. However, some participants took longer to familiarise themselves with the questions displayed and explore available features.

The AI classification system was seen as mostly accurate, with users appreciating the automation in categorising and prioritising questions. Regardless, some users noted occasional inconsistencies or a lack of clear distinction between different classification types, such as requirements versus architectural decisions.

The classification system contributed to managing the backlog effectively, and the detection of semantically similar questions indicated by the participants assured that the embedding model was performing as expected.

Task 2 – Manage the Board

This task saw slightly briefer completion times ranging from one to three minutes. According to most participants, the board interface for managing question status and priority was intuitive

and easy to navigate. Search and filter functionalities helped users discover relevant questions efficiently, though some suggested some usability improvements such as clearer priority selectors and better visual cues (e.g., colour coding or icons).

Participants were generally confident using the different roles available to modify question attributes. However, a few mentioned some initial confusion about the distinctions between roles like Decision Maker and Product Owner. Overall, the quick completion times and constructive feedback indicate that while the board interface is already effective, it still holds potential for further usability enhancements.

Task 3 – Discuss and Resolve

Participants commonly spent between 2 and 5 minutes on this task, with some exploring this task for over 10 minutes. The variation depended on how deeply they engaged with the AI assistant and how confident they felt making decisions. The majority of participants indicated that the AI assistant supported their reasoning process and played a role in shaping the final decision regarding the status of each question. Users who spent more time often appreciated the AI's ability to "reason with them", while others found the answers to be straight to the point.

One software architect expert, when discussing an issue with the AI, found that the assistant mimicked human collaboration rituals not suitable for an AI assistant, such as asking to "schedule a follow-up to discuss potential scalability challenges as we progress" Nonetheless, the feature was valued for stimulating thought and aiding in the reasoning process and was also deemed helpful when resolving an issue.

Reflection Phase

Overall, users found the tool intuitive and valuable, with the board view for visualising questions and the AI assistant discussion feature frequently mentioned as the most valuable features. Limitations raised included some interface usability concerns, such as the need for clearer action buttons, better colour differentiation and the desire for the interface to be more lightweight. Several participants suggested improvements around AI behaviour, preferring it to act more like an assistant than a conversational partner and wished for more transparency in AI suggestions. All respondents expressed willingness to consider using such a tool in their professional work, highlighting the potential impact on managing software development discussions and decisions.

Lastly, while there is room for refinement, particularly in the AI's refinement and interface clarity, the overwhelmingly positive reception suggests strong potential for improving and integrating such a tool into real-world software development workflows.

Chapter 6

Conclusions

This dissertation has demonstrated how systematically eliciting, organising and tracking natural questions during software architecture design can effectively support decision-making and reduce the probability of knowledge loss. Natural questions and architectural knowledge are frequently fragmented and insufficiently preserved, even though they are crucial for scalable and sustainable software systems. By interpreting questions as dynamic architectural entities, we aimed to explore a solution to foster a more thoughtful and traceable design practice where the questions and rationale behind decisions remain accessible throughout the project's development.

The first obstacle in this research was to understand how necessary the development of a tool to support this process was, and by exploring and comparing current tools and traditional methods, it was confirmed that the current means were insufficient for a task of this importance. Current methods lack the appropriate structure due to not being tailored explicitly for architectural knowledge management and question management, and traditional methods lack the structure and rigour needed to support long-term knowledge retention and retrieval.

Given this identified gap, in this thesis, we focused on designing and validating a dedicated tool that facilitates the structured handling of architectural questions, through an extensive literature review, an exploratory case study and two surveys with experts and software architecture university students. Throughout this process, we formalised and validated a representation of the lifecycle of questions, focusing on the roles, phases and decision points involved in their evolution, ensuring that the architectural questions are contextualised and preserved. Understanding the nature of the questions and the roles involved, expert feedback was key to eliciting and validating some functional and non-functional requirements to ensure the tool's applicability, usability, and effectiveness in real-world architectural settings.

Building on these foundations, we designed and prototyped a tool that is grounded and validated through the lifecycle and requirements before its establishment, delivering a tool to support the capture, organisation, and management of architectural questions throughout the design process. Unlike existing tools and methods that capture architectural documentation statically, our tool focuses on the dynamic nature of the questions that drive decisions. This tool focused on leveraging GenAI technologies and capabilities to automate and improve classification, identify

similar questions and enhance decision-making by supporting reflective dialogue.

Conducting the research in this thesis highlighted the overlooked complexity of capturing and managing questions and the damage it can cause in real-time design contexts. Knowledge loss remains costly, undermining software systems' quality, maintainability and evolution. I hope this work inspires researchers and practitioners to elevate questions as core entities within the architectural process and adopt more structured, specialised approaches to knowledge preservation. In doing so, reducing knowledge loss over time and cultivating more knowledge-aware and sustainable software architecture practices is possible.

6.1 Summary of Contributions

Throughout the exploratory process, we identified gaps in knowledge relating to the natural questions that emerge in software architecture. Given these gaps, with a relevant body of literature, exploratory case studies, surveys and a prototype, the primary focus was contributing to this relatively unexplored topic.

The key contributions can be categorised in four different areas, theoretical, methodological, empirical and technical:

- **Theoretical:** We defined our perception of a lifecycle for managing natural questions in software architecture, mapping the lifecycle phases and roles based on our findings.
- **Methodological:** Followed a structured research process, combining design science, literature review and expert feedback.
- **Empirical:** Validated the lifecycle phases, roles and features needed for a tool to manage them by designing a workshop and a survey with expert architects and engineers.
- **Technical:** Developed an AI-enhanced question tracking prototype that integrates questions' backlog management, enables discussions with AI and between team members and features AI categorisation for question management.

6.2 Answers to Research Questions

At the beginning of this thesis, we identified three research questions as the most prominent ones to investigate the potential of GenAI technologies to support the preservation, elicitation and management of knowledge derived from questions. The following is a collection of those three questions and a reflection on each based on the outcomes of this research:

RQ1: Are the traditional methods insufficient for retaining knowledge conveniently in all phases, thereby justifying a need to integrate GenAI technologies?

The simple answer is yes. Throughout the exploratory process, from the literature review to the surveys and exploratory case study, traditional methods, such as notes, post-its, static

documentation and informal conversations, lack the structure and capability to capture architectural questions across all decision-making phases. These approaches may seem sufficient when capturing questions and thoughts initially, but they cannot preserve rationale and are one of the leading causes of knowledge loss over time. These findings justify the integration of GenAI technologies, which can adapt, anticipate and provide dynamic, context-aware support for organising, retrieving and preserving such questions and their inherent knowledge.

RQ2: How successfully can GenAI technologies be used to anticipate the repercussions of each decision and aid the process of decision-making to be more effective?
While developing and evaluating this tool, GenAI technologies showed promising potential in assisting architectural decision-making. Being able to anticipate the consequences of any question and decision remains complex. Still, the AI-supported tool provided practical and relevant responses, helping architects reflect more critically on downsides and potential advantages with each decision. This is further supported by the findings of the tool's evaluation with experts and university students, who, while trying the tool, found that the AI assistant provided helpful feedback and contributed to their overall decision on specific questions. The prototype demonstrated that GenAI can increase decision clarity and contribute immensely to more effective and informed architectural decisions.

RQ3: To what extent can a specialised tool contribute to structuring the management of architectural questions throughout the software design process?
This thesis accentuates a specialised tool's substantial contribution to structuring the management of architectural questions. Following a lifecycle-oriented approach, the tool enables the systematic capture and organisation of questions that are often lost or fragmented in traditional workflows, given their dynamic nature.

The prototype developed in this thesis demonstrates how such a tool can improve traceability, reduce redundancy, and facilitate context-aware decision making. Feedback from expert software architects validated the lifecycle and role design and highlighted the tool's potential to address a gap in architectural knowledge preservation and decision-making.

While, as stated previously, this tool is still in a research-stage implementation, the AI support combined with the integration of structured processes suggests a promising path for improving knowledge preservation and decision support in architectural practice.

6.3 Limitations

As with any research endeavour, primarily a tool, this work can be improved even further, but it presents some limitations. Acknowledging these limitations to set realistic expectations and provide context when interpreting results is essential.

- **Prototype:** The current prototype remains a research prototype and is unsuitable for production environments. Its features and interfaces can be built upon, but are designed primarily for research and validation.
- **Evaluation scope:** The tool was evaluated with a group of software architects and context-aware university students. While their feedback was worthwhile, there is a need for broader testing within different project contexts to capture and understand the findings better.
- **AI service dependency:** A key "cog" in this tool is AI integration. This tool relies on Google's Gemini models APIs, which may limit long-term maintainability, performance consistency and scalability, due to strict limits for embedding that can only be improved with a significant monetary investment.
- **Lack of longitudinal study:** The tool has not yet been tested in real-world project settings. There is a clear need to evaluate the effectiveness of supporting architectural knowledge over extended periods to determine its long-term impact, adoption, and sustainability in real-world projects.

6.4 Future Work

While this work has contributed to the gap in natural questions management, several improvements can and should be further explored to build upon the foundation laid for a future with less knowledge loss and more knowledge preservation. The following steps for future research should aim to broaden its applicability and practical utility while also fostering a wider use of the tool. To this end, future work can explore these suggestions to enhance the tool's effectiveness, scalability, and integration into real-world software architecture practices:

- **Evaluation in a Real-World Setting** - Deploy and assess the tool in industry projects over three to six months to gather real usage data and feedback.
- **Improve AI integration** - AI keeps evolving at an incredible pace, and with it, this tool should also improve. Future iterations could incorporate AI assistants that offer more proactive, context-aware support throughout the lifecycle of architectural questions, leveraging full-context awareness for each project, allowing for improved feedback and management.
- **Integration with Popular Tools** - Explore integrating the tool capabilities with popular tools, such as GitHub, Jira and VSCode, to seamlessly embed question management into existing software engineering workflows.
- **Visualisation Improvement** - Explore the idea revolving around one of the prototypes where the questions are visualised as a network with clusters of questions and connections between related ones.

- **Build upon App Capabilities** - While the app currently offers an easy way to capture questions through speech-to-text, build upon it to record and store full conversations and thoughts during meetings or reviews while the AI trims down the unnecessary information and preserves the questions and rationale.

6.5 Final Remarks

This thesis has been deeply enlightening despite being intellectually demanding. What started as an exploration into improving software design processes revealed a profound and underexplored complexity: managing natural questions and their lifecycle. I was astounded by how important these questions are for architectural knowledge and reasoning, and how often they go undocumented and are lost over time.

One of the most challenging aspects was conceptualising a question's lifecycle when these are so dynamic and can emerge spontaneously at any stage of the design process, making them difficult to anticipate or categorise. It led to a deeper understanding of the importance of human factors in times that seem to shift away from human involvement and increasingly rely on artificial intelligence. AI is beneficial and has been used throughout the development of tools to aid human intervention, not replace it. Taking the most of AI is essential and should be cultivated with rigour and a clear ethical framework to ensure it augments rather than diminishes human creativity.

This dissertation has reaffirmed the critical role of questions in shaping architectural knowledge. Treating them as first-class entities opens new possibilities for supporting more knowledge-aware and reflective practices in software architecture.

Finally, I wish this work serves as a basis for future research and practice that emphasises the "whys" behind every decision, not just outcomes, for it is in the reasoning, not just the result, where actual architectural knowledge resides.

Bibliography

[1] R.R. Althar and D. Samanta. "Application of Machine Intelligence-Based Knowledge Graphs for Software Engineering". In: 2021. URL: https://doi.org/10.4018/978-1-7998-7701-1.ch010.

[2] Preethu Rose Anish. "Towards an Approach to Stimulate Architectural Thinking during Requirements Engineering Phase". In: *2016 IEEE 24th International Requirements Engineering Conference (RE)*. 2016, pp. 421–426. DOI: 10.1109/RE.2016.30.

[3] Z. Anwar et al. "Collaborative Solutions to Software Architecture Challenges Faced by IT Professionals". In: 2024. URL: https://doi.org/10.4018/IJHCITP.342839.

[4] Behzad Bastani. "A requirements analysis framework for open systems requirements engineering". In: *SIGSOFT Softw. Eng. Notes* 32.2 (Mar. 2007), pp. 1–19. ISSN: 0163-5948. DOI: 10.1145/1234741.1234753. URL: https://doi.org/10.1145/1234741.1234753.

[5] G. Borrego, S. González-López, and R.R. Palacio. "Tags Recommender to Classify Architectural Knowledge Applying Language Models". In: 2022. URL: https://doi.org/10.3390/math10030446.

[6] Jan vom Brocke, Alan Hevner, and Alexander Maedche. "Introduction to Design Science Research". In: *Design Science Research. Cases*. Ed. by Jan vom Brocke, Alan Hevner, and Alexander Maedche. Cham: Springer International Publishing, 2020, pp. 1–13. DOI: 10.1007/978-3-030-46781-4_1. URL: https://doi.org/10.1007/978-3-030-46781-4_1.

[7] Lianping Chen, Muhammad Ali Babar, and Bashar Nuseibeh. "Characterizing Architecturally Significant Requirements". In: *IEEE Software* 30.2 (2013), pp. 38–45. DOI: 10.1109/MS.2012.174.

[8] Douglas Cumming et al. "Towards AI Ethics-Led Sustainability Frameworks and Toolkits: Review and Research Agenda". In: 2024. URL: https://doi.org/10.1016/j.josfa.2024.100003.

[9] Jose Oriol Lopez Berengueres Danielle Drozdzewski. "Developing QualNotes: A collaborative and cross-disciplinary ethnography". In: 2024. URL: https://doi.org/10.1016/j.diggeo.2024.100086.

[10] R. Dhar, K. Vaidhyanathan, and V. Varma. "Can LLMs Generate Architectural Design Decisions? - An Exploratory Empirical Study". In: 2024. URL: https://doi.org/10.1109/ICSA59870.2024.00016.

[11] R. Dhar, K. Vaidhyanathan, and V. Varma. "Leveraging Generative AI for Architecture Knowledge Management". In: 2024. URL: https://doi.org/10.1109/ICSA-C63560.2024.00034.

[12] F. Gilson and D. Weyns. "When Natural Language Processing Jumps into Collaborative Software Engineering." In: 2019. URL: https://doi.org/10.1109/ICSA-C.2019.00049.

[13] Ryan C. Godwin et al. "IRB-Draft-Generator: A Generative AI Tool to Streamline the Creation of Institutional Review Board Applications". In: 2024. URL: https://doi.org/10.1016/j.softx.2023.101601.

[14] Sebastian Hahner et al. "ARC3N: A Collaborative Uncertainty Catalog to Address the Awareness Problem of Model-Based Confidentiality Analysis." In: 2024. URL: https://doi.org/10.1145/3652620.3688556.

[15] T Händler. "A Taxonomy for Autonomous LLM-Powered Multi-Agent Architectures". In: 2023. URL: doi.org/10.5220/0012239100003598.

[16] N. Harrison and A. Aguiar. "The Nature of Questions that Arise During Software Architecture". In: *2024 European Conference on Software Architecture (ECSA)*. 2024. URL: https://conf.researchr.org/details/ecsa-2024/ecsa-2024-research-papers/2/The-Nature-of-Questions-that-Arise-During-Software-Architecture.

[17] Neil B. Harrison et al. "Capturing notable architectural decisions". In: *2023 IEEE 20th International Conference on Software Architecture Companion (ICSA-C)*. 2023, pp. 179–182. DOI: 10.1109/ICSA-C57050.2023.00047.

[18] Anton Jansen and Jan Bosch. "Software Architecture as a Set of Architectural Design Decisions". In: *Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture*. WICSA '05. USA: IEEE Computer Society, 2005, pp. 109–120. ISBN: 0769525482. DOI: 10.1109/WICSA.2005.61. URL: https://doi.org/10.1109/WICSA.2005.61.

[19] Rick Kazman and Mark Klein. "Performing architecture tradeoff analysis". In: *Proceedings of the Third International Workshop on Software Architecture*. ISAW '98. Orlando, Florida, USA: Association for Computing Machinery, 1998, pp. 85–88. ISBN: 1581130813. DOI: 10.1145/288408.288430. URL: https://doi.org/10.1145/288408.288430.

[20] P Kokol. "The Use of AI in Software Engineering: A Synthetic Knowledge Synthesis of the Recent Research Literature". In: 2024. URL: https://doi.org/10.3390/info15060354.

[21] Carmelo Fabio Longo et al. "A Framework for Cognitive Chatbots Based on Abductive–Deductive Inference." In: 2023. URL: https://doi.org/10.1016/j.cogsys.2023.05.002.

[22] J. Albert Mills, Gabrielle Durepos, and Elden Wiebe. "Exploratory Case Study". In: *Encyclopedia of Case Study Research*. SAGE Publications, Inc., 2010, pp. 372–373. DOI: 10.4135/9781412957397. URL: https://doi.org/10.4135/9781412957397.

[23] Christine Miyachi. "Agile software architecture". In: *SIGSOFT Softw. Eng. Notes* 36.2 (Mar. 2011), pp. 1–3. ISSN: 0163-5948. DOI: 10.1145/1943371.1943388. URL: https://doi.org/10.1145/1943371.1943388.

[24] Ikujiro Nonaka and Noboru Konno. "The Concept of 'Ba': Building a Foundation for Knowledge Creation". In: 1998. URL: https://home.business.utah.edu/actme/7410/nonaka%201998.pdf.

[25] Robert L. Nord, Nanette Brown, and Ipek Ozkaya. "Architecting with just enough information". In: *Proceedings of the 6th International Workshop on SHAring and Reusing Architectural Knowledge*. SHARK '11. Waikiki, Honolulu, HI, USA: Association for Computing Machinery, 2011, pp. 9–12. ISBN: 9781450305969. DOI: 10.1145/1988676.1988680. URL: https://doi.org/10.1145/1988676.1988680.

[26] Sharmila Reddy Pappula and Sathwik Rao Allam. "LLMs for Conversational AI: Enhancing Chatbots and Virtual Assistants". In: *International Journal of Research Publication and Reviews* 4.12 (Dec. 2023), pp. 1601–1611. ISSN: 2582-7421. DOI: 10.55248/gengpi.4.1223.123425. URL: https://doi.org/10.55248/gengpi.4.1223.123425.

[27] Kumaran Rajaram and Patrick Nicolas Tinguely. "Generative Artificial Intelligence in Small and Medium Enterprises: Navigating Its Promises and Challenges". In: 2024. URL: https://doi.org/10.1016/j.bushor.2024.05.008.

[28] Inc. Rayyan Systems. *Rayyan - Intelligent Systematic Review*. https://www.rayyan.ai/. 2024.

[29] Steven I. Ross et al. "The Programmer's Assistant: Conversational Interaction with a Large Language Model for Software Development". In: *IUI '23: 28th International Conference on Intelligent User Interfaces*. Association for Computing Machinery. Sydney, NSW, Australia: ACM, 2023. DOI: 10.1145/3581641.3584037. URL: https://doi.org/10.1145/3581641.3584037.

[30] S.A. Rukmono, L. Ochoa, and M.R.V. Chaudron. "Deductive Software Architecture Recovery via Chain-of-Thought Prompting". In: 2024. URL: https://doi.org/10.1145/3639476.3639776.

[31] Ayyappa Sajja, Dheerender Thakur, and Aditya Mehra. "Integrating Generative AI into the Software Development Lifecycle: Impacts on Code Quality and Maintenance". In: *International Journal of Science and Research Archive* 13.1 (Oct. 2024), pp. 1952–1960. DOI: 10.30574/ijsra.2024.13.1.1837.

[32] Rounok Salehin. "Missing Requirements Information and its Impact on Software Architectures: A Case Study". In: *Masters Thesis, University of Western Ontario*. London, Ontario, Canada, 2013. URL: https://api.semanticscholar.org/CorpusID:28003460.

[33] Dachuan Shi, Philipp Liedl, and Thomas Bauernhansl. “Interoperable Information Modelling Leveraging Asset Administration Shell and Large Language Model for Quality Control toward Zero Defect Manufacturing”. In: 2024. URL: https://doi.org/10.1016/j.jmsy.2024.10.011.

[34] Ian Sommerville and Jane Ransom. “An empirical study of industrial requirements engineering process assessment and improvement”. In: *ACM Trans. Softw. Eng. Methodol.* 14.1 (Jan. 2005), pp. 85–117. ISSN: 1049-331X. DOI: 10.1145/1044834.1044837. URL: https://doi.org/10.1145/1044834.1044837.

[35] Streamlit Inc. *Streamlit*. https://streamlit.io. 2023.

[36] Ricky. Sun. “Chapter 2 - Graph Database Basics and Principles”. In: 2024. URL: https://doi.org/10.1016/B978-0-443-14162-1.00001-5.

[37] Jeff Sutherland et al. *A Scrum Book: The Spirit of the Game*. 1st ed. Pragmatic Programmers. Pragmatic Bookshelf, Aug. 2019. ISBN: 978-1-68050-671-6. URL: https://pragprog.com/book/jcscrum/a-scrum-book.

[38] Antony Tang et al. “A Comparative Study of Architecture Knowledge Management Tools”. In: *Journal of Systems and Software* 83.3 (2010), pp. 352–370. DOI: 10.1016/j.jss.2009.08.032.

[39] Parcifal Team. *Parcifal*. https://parsif.al/. 2024.

[40] United Nations. *Sustainable Development Goals*. https://sdgs.un.org/goals. Accessed: 2025-06-28. 2025.

[41] Dimitri Van Landuyt, Eddy Truyen, and Wouter Joosen. “On the modularity impact of architectural assumptions”. In: *Proceedings of the 2012 Workshop on Next Generation Modularity Approaches for Requirements and Architecture*. NEMARA ’12. Potsdam, Germany: Association for Computing Machinery, 2012, pp. 13–16. ISBN: 9781450311274. DOI: 10.1145/2162004.2162008. URL: https://doi.org/10.1145/2162004.2162008.

[42] Chen Yang. “Architectural assumptions and their management in software development”. Publisher’s PDF, University of Groningen Research Database. PhD thesis. Groningen, The Netherlands: University of Groningen, 2018. URL: https://www.rug.nl/research/portal/publications/architectural-assumptions-and-their-management-in-software-development(52c75a4c-bff9-4629-b13a-e001e917dfbb).html.

Appendix A

Repository

The prototype developed as part of this dissertation is publicly available on GitHub at the following link:

https://github.com/up202003484/questionManagement

This repository contains the full implementation of the architectural question management tool described in the thesis, as well as the question text-to-speech app code and the receiver source code.

Appendix B

Prompts

Classification Prompt: Category + Priority + Taxonomy

You are an expert in software architecture.

Given the question below:
"<QUESTION_TEXT>"

1. Suggest the most relevant **category** from this list: [architecture, performance, security, integration, maintainability, testing, devops].
Respond only with the category name and with an upper first letter.

2. Suggest the most relevant **priority** from this list: [Low, Medium, High].
Use this scale:
- 1-4 → Low
- 5-7 → Medium
- 8-10 → High
High = blocking/urgent.
Low = non-blocking/postpone.
Medium = important but not blocking.
Respond only with the priority name, no numbers nor special characters.

3. Tag the question with 1 to 3 relevant **taxonomy tags** from this list or others you find more fitting:

"*ArchitecturalStyle*","*QualityAttribute*","*DesignDecision*","*Communication*","*Process*","*Deployment*","

```
Respond strictly in the format:
Taxonomy:  <comma-separated list of tags>

--
Now, classify:
```

AI Response Prompt: Product Owner

```
Act as an helpful assistant who has the knowledge of a Product
Owner.  Focus on business value, stakeholder alignment, and requirements
clarity.

Respond with a message that fits naturally into the ongoing discussion
and try to act like a normal person acting as a product owner
in a company.

Ensure your response is clear, actionable, and directly addresses
the core concern.

Respond only with the message and be brief and concise, don't
add any extra unnecessary information or context.

If necessary, provide any trade-offs, benefits, or potential
challenges related to the question at hand.
```

AI Response Prompt: Developer

```
Act as a helpful assistant who has the knowledge of a Developer.
Focus on implementation details, trade-offs, and feasibility.

Respond with a message that fits naturally into the ongoing discussion
and try to act like a normal person acting as a developer in
a company, but still behave professionally.
```

```
Ensure your response is clear, actionable, and directly addresses
the core concern.

Respond only with the message and be brief and concise, don't
add any extra unnecessary information or context.

If necessary, provide any trade-offs, benefits, or potential
challenges related to the question at hand.
```

AI Response Prompt: Question Owner

```
Act as an helpful assistant who has the knowledge of a Software
Architect and you act as a Questions Owner assistant.  A questions
owner formulates the question and introduces it into the workflow,
discussing it with other members to ensure it is clearly defined
while also managing the question backlog and archival process.

Focus on architectural quality attributes, trade-offs, and long-term
implications.

Respond with a message that fits naturally into the ongoing discussion
and try to act like a normal person acting as a questions owner
in a company.

Ensure your response is clear, actionable, and directly addresses
the core concern.

Respond only with the message and be brief and concise, don't
add any extra unnecessary information or context.

If necessary, provide any trade-offs, benefits, or potential
challenges related to the question at hand.
```

AI Response Prompt: Decision Maker

```
Act as a helpful assistant who has the knowledge of a Decision
Maker.  Focus on strategic direction, risk, and prioritisation.
A decision maker evaluates all parameters and determines whether
```

```
the question is resolved, deferred, or answered through assumptions
so you need to have good feedback that can impact the end decision.

Respond with a message that fits naturally into the ongoing discussion
and try to act like a normal person acting as a decision maker
in a company.

Ensure your response is clear, actionable, and directly addresses
the core concern.

Respond only with the message and be brief and concise, don't
add any extra unnecessary information or context.

If necessary, provide any trade-offs, benefits, or potential
challenges related to the question at hand.
```