A DECISION SUPPORT SYSTEM FOR SOFTWARE ARCHITECTURE DECISION MAKING

YAZILIM MİMARİSİNE KARAR VERME İÇİN BİR KARAR DESTEK SİSTEMİ

MERVE ÖZDEŞ DEMİR

ASSOC. PROF. DR. AYÇA KOLUKISA TARHAN

Supervisor

ASSOC. PROF. DR. OUMOUT CHOUSEINOGLOU

2nd Supervisor

Submitted to

Graduate School of Science and Engineering of Hacettepe University

as a Partial Fulfillment to the Requirements

for the Award of the Degree of Doctor of Philosophy

in Computer Engineering

June 2024

ABSTRACT

A DECISION SUPPORT SYSTEM FOR SOFTWARE ARCHITECTURE DECISION MAKING

Merve Özdeş Demir

Doctor of Philosophy, Computer Engineering

Supervisor: Assoc. Prof. Dr. Ayça Kolukısa Tarhan

2nd Supervisor: Assoc. Prof. Dr. Oumout Chouseinoglou

June 2024, 193 pages

Software architecture forms the overall structure of a software system and defines how its components work and interact together. It serves as a critical schema that specifies in detail the structure, elements and communication channels of the system. A well designed architecture ensures that new functionality is seamlessly integrated, changes are adapted to, and ongoing efficiency is maintained, making the software scalable, maintainable, and robust. This is particularly important in large-scale projects to manage complexity and align the software's technical specifications with business goals. In the rapidly evolving world of software engineering, the difficulty of decision-making in the architectural decision making process is a major challenge and affects the quality, maintainability and scalability of the system. In response to this challenge, this thesis introduces a tool titled SOFtware ARchitecture Decision Support System (SOFAR-DSS), specifically designed to support more structured decision making in the software architecture design process. SOFAR-DSS is the culmination of an exploratory study and an extensive survey designed to identify key factors and common challenges within this crucial process. The initial stages of this thesis involved a meticulous exploration to pinpoint the elements influencing architectural

decision-making and to delineate the hurdles typically encountered. This study began with semi-structured interviews conducted online with nine experts, providing valuable insights that laid the groundwork for the subsequent survey. The exploratory study, enriched by feedback from these professionals, set the stage for a comprehensive survey aimed at a broader audience to scrutinize the architectural decision-making process in greater detail. The survey reached 101 participants from various countries, encompassing architects with varying years of experience, thus ensuring a rich diversity of insights and perspectives. The insights garnered from these preliminary studies have been instrumental in shaping the development of the decision support system. SOFAR-DSS serves as a support system that guides decision makers through the complexity of architectural decisions. While its primary target is software architect, through its intuitive design and comprehensive guidance, SOFAR-DSS addresses a wider audience of professionals involved in the software architecture decision process. The primary aim of SOFAR-DSS is to refine the decision-making process, guaranteeing that architectural decisions enhance the project's needs, thus not only improving the efficiency of the decision-making process but also contributing to the overall quality and uniformity of the software architecture. SOFAR-DSS is differentiated from other decision support systems through its integration of artificial intelligence and an interactive interface, providing decision-makers with more efficient, user-centric, and superior guidance. In order to validate SOFAR-DSS, we employed three different approaches. First, we tested the system by inputting design pattern definitions from books and assessing its accuracy and relevance. Second, we benchmarked the system's performance by entering real-world questions and answers from Stack Overflow, comparing the system's responses to those given by the community. Finally, we conducted hands-on trials with five experienced experts who thoroughly evaluated the system's effectiveness in practical scenarios. The system has successfully generated effective responses for sample issues tested across a range of scenarios.

Keywords: software architecture, decision support system, ontology, software desing pattern.

ÖZET

YAZILIM MİMARİSİNE KARAR VERME İÇİN BİR KARAR DESTEK SİSTEMİ

Merve Özdeş Demir

Doktora, Bilgisayar Mühendisliği

Danışman: Doç. Dr. Ayça Kolukısa Tarhan

Eş Danışman: Doç. Dr. Oumout Chouseinoglou

Haziran 2024, 193 sayfa

Yazılım mimarisi, bir yazılım sisteminin genel yapısını oluşturur ve bileşenlerinin birlikte nasıl çalıştığını ve etkileşimde bulunduğunu tanımlar. Sistemin yapısını, unsurlarını ve iletişim kanallarını ayrıntılı olarak belirten kritik bir şema görevi görür. İyi tasarlanmış bir mimari, yeni işlevlerin sorunsuz bir şekilde entegre edilmesini, değişikliklere uyum sağlanmasını ve süregelen verimliliğin korunmasını sağlayarak yazılımı ölçeklenebilir, sürdürülebilir ve sağlam hale getirir. Bu özellikle büyük ölçekli projelerde karmaşıklığı yönetmek ve yazılımın teknik özelliklerini iş hedefleriyle uyumlu hale getirmek için önemlidir. Yazılım mühendisliğinin hızla gelişen dünyasında, mimari karar verme sürecinde karar vermenin zorluğu önemli bir sorundur ve sistemin kalitesini, sürdürülebilirliğini ve ölçeklenebilirliğini etkiler. Bu tez, bu zorluğa yanıt olarak, yazılım geliştirme sürecinde daha yapılandırılmış karar vermeyi desteklemek için özel olarak tasarlanmış yazılım mimarisi karar destek sistemi (SOFAR-DSS) adlı bir aracı tanıtmaktadır. SOFAR-DSS, bu önemli süreçteki temel faktörleri ve ortak zorlukları belirlemek için tasarlanmış kapsamlı bir anket çalışmasının ve keşifsel bir çalışmanın sonucudur. Bu tezin ilk aşamaları, mimari karar verme sürecini etkileyen unsurları belirlemek ve tipik olarak karşılaşılan engelleri

tanımlamak için titiz bir araştırmayı içeriyordu. Bu çalışma, dokuz uzmanla çevrimiçi olarak gerçekleştirilen yarı yapılandırılmış görüşmelerle başlamış ve sonraki anket için zemin hazırlayan değerli bilgiler sağlamıştır. Uzmanlardan alınan geri bildirimlerle zenginleştirilen keşif çalışması, mimari karar verme sürecini daha ayrıntılı bir şekilde incelemek için daha geniş bir kitleyi hedefleyen kapsamlı bir anket için zemin hazırladı. Anket, çeşitli ülkelerden, farklı deneyim yıllarına sahip mimarları kapsayan 101 katılımcıya ulaşmış ve böylece zengin bir içgörü ve bakış açısı çeşitliliği sağlamıştır. Bu ön çalışmalardan elde edilen içgörüler SOFAR-DSS'nin gelişimini şekillendirmede etkili olmuştur. SOFAR-DSS, mimari kararların karmaşıklığında karar vericilere rehberlik eden bir destek olarak hizmet vermektedir. Birincil hedefi yazılım mimarları olsa da, SOFAR-DSS sezgisel tasarımı ve kapsamlı rehberliği sayesinde yazılım mimarisi karar sürecine dahil olan daha geniş bir profesyonel kitleye hitap etmektedir. SOFAR-DSS'nin birincil amacı, mimari kararların projenin ihtiyaçlarını karşılamasını garanti ederek karar verme sürecini iyileştirmek ve böylece yalnızca karar verme sürecinin verimliliğini artırmak değil, aynı zamanda yazılım mimarisinin genel kalitesine ve bütünlüğüne katkıda bulunmaktır. SOFAR-DSS, yapay zeka ve etkileşimli arayüz entegrasyonu sayesinde diğer karar destek sistemlerinden farklılaşmakta ve karar vericilere daha verimli, kullanıcı merkezli ve üstün bir rehberlik sağlamaktadır. SOFAR-DSS'yi doğrulamak için üç farklı yaklaşım benimsedik. İlk olarak, sistemin doğruluğunu ve alaka düzeyini değerlendirerek, kitaplardan alınan tasarım deseni tanımlarını sisteme girdik. İkinci olarak, Stack Overflow'dan alınan gerçek dünya soruları ve cevaplarını sisteme girerek, sistemin performansını topluluğun verdiği yanıtlarla karşılaştırdık. Son olarak, beş deneyimli uzmanın sistemi pratik senaryolarda kapsamlı bir şekilde değerlendirdiği uygulamalı denemeler gerçekleştirdik. Sistem, çeşitli senaryolarda test edilen örnek sorunlar için başarılı bir şekilde etkili yanıtlar üretmiştir.

Keywords: yazılım mimarisi, karar destek sistemi, ontoloji, yazılım tasarım örüntüsü.

ACKNOWLEDGEMENTS

I am grateful to my thesis advisor Assoc. Prof. Dr. Ayça Kolukısa Tarhan for her expertise, knowledge, contributions and support throughout the thesis process. I would also like to express my sincere gratitude to my thesis co-supervisor Assoc. Prof. Dr. Oumout Chouseinoglu, whose expertise, understanding, patience and invaluable support contributed significantly to my PhD experience. His perspective and insights contributed significantly to the depth of this study.

I would also like to express my sincere thanks to my jury members Prof. Dr. Altan Koçyiğit and Prof. Dr. Ebru Akçapınar Sezer for their constructive feedback and important suggestions throughout this thesis process. Their detailed questions and interactions with my work were instrumental in improving the quality and coherence of my research.

I cannot forget the unwavering support and motivation of Dr. Necva Bölücü, Dr. Nebi Yılmaz, Dr. Bahar Gezici Geçer, and Burçak Asal, who have been with me through the ups and downs of this academic journey. Their belief in me and constant encouragement has been my source of strength and resilience.

Above all, I owe my deepest gratitude to my family and my husband, who have given me endless love, support and encouragement throughout this process. Their sacrifices, understanding and belief in my abilities have been the backbone of my perseverance and success. The journey to complete this thesis was made possible not only by my efforts, but also by the collective support of these extraordinary individuals.

I am eternally grateful to all of you. Your encouragement and support have profoundly shaped this academic work and my personal development along the way.

CONTENTS

Page

TABLES

FIGURES

ABBREVIATIONS

ADD	:	**A**rchitectural **D**esign **D**ecision
AHP	:	**A**nalytical **H**ierarchy **P**rocess
AI	:	**A**rtificial **I**ntelligence
AK	:	**A**rchitectural **K**nowledge
ATAM	:	**A**rchitecture **T**radeoff **A**nalysis **M**ethod
BERT	:	**B**idirectional **E**ncoder **R**epresentations from **T**ransformers
BoW	:	**B**ag **o**f **W**ords
CBAM	:	**C**ost **B**enefit **A**nalysis **M**ethod
DSS	:	**D**ecision **S**upport **S**ystem
HAM	:	**H**ybrid **A**ssessment **M**ethod
IDSS	:	**I**ntelligent **D**ecision **S**upport **S**ystem
KNN	:	**K**-**N**earest **N**eighbours
LGBM	:	**L**ight **G**radient **B**oosting **M**achine
LSTM	:	**L**ong **S**hort **T**erm **M**emory
MCDM	:	**M**ulti **C**riteria **D**ecision **M**aking
NLP	:	**N**atural **L**anguage **P**rocessing
PoC	:	**P**roof **o**f **C**oncept
RF	:	**R**andom **F**orest
QA	:	**Q**uestion **A**nswering
SDLC	:	**S**oftware **D**evelopment **L**ife **C**ycle
SOFAR-DSS	:	**SOF**tware **AR**chitecture **D**ecision **S**upport **S**ystem
SVM	:	**S**upport **V**ector **M**achines
TF-IDF	:	**T**erm **F**requency-**I**nverse **D**ocument **F**requency
XAI	:	E**X**plainable **A**rtifical **I**ntelligence
XGBoost	:	E**X**treme **G**radient **Boost**ing

1. INTRODUCTION

The field of software development is constantly evolving, driven by the increasing demand for robust, scalable, and agile applications and advancements in the field of technology. At the heart of developing effective and efficient software systems lies the critical process of making architectural decisions [1]. These decisions significantly influence the system's capabilities and adaptability, necessitating a deep understanding of various architectural patterns and practices tailored to specific project needs [2]. As software systems grow in size and complexity, managing this complexity and ensuring system integrity becomes increasingly challenging. This necessitates steering the project in accordance with a specific software architecture [3]. While there are many interpretations of what software architecture entails, the definition by Bass et al. [3] is the most widely recognized, describing it as the system's essential structures, encompassing software components, their interrelations, and characteristics.

In software architecture, decision-making involves a complex array of choices, each with substantial impact on aspects like quality, performance, and maintainability of the software [1]. This process is full of challenges that require architects to balance technical constraints, business objectives, scalability, sustainability, and ever-present time and budget constraints [4]. Software system design is comprised of multiple critical decisions [5]. These decisions, traditionally seen as significant choices about the system [6], involve evaluating different alternatives and selecting the most suitable one. The breadth and complexity of these alternatives can be daunting, yet their understanding is essential for impactful decision-making that directly influences system quality. Primarily, software architects are responsible for these architectural decisions [7], [8], but involving various stakeholders can enable architectural requirements to be addressed from more diverse perspectives.

The complexity of decision-making in software architecture has led to the development of various methods and procedures designed to assist architects. Notable among these are the Architecture Tradeoff Analysis Method (ATAM) [9], the Cost-Benefit Analysis

Method (CBAM) [10], the Quality-Driven Decision Support Method [11], and ATRIUM [12]. However, these methods often lack automated tools for analyzing decision models, or when such tools exist, they rely on predetermined models and equations, focusing more on decision outcomes rather than the decision-making process itself [13].

The integration of artificial intelligence (AI) technologies across the various stages of the software engineering lifecycle, particularly in architectural decision-making, represents a significant evolutionary step in the discipline. This integration extends beyond mere efficiency improvements, encompassing a transformative effect on the entire process, from initial project planning to maintenance. As detailed in the study employing a mixed-method approach, AI applications are critically influencing each phase of software development, including problem analysis, design, implementation, and testing [14]. Furthermore, case studies such as IBM's Watson and Google's AlphaGo highlight AI's capability to revolutionize complex problem-solving and decision-making processes in software engineering, demonstrating the profound impact of AI technologies in enhancing understanding and optimizing solutions [15]. Additionally, the advancements in machine learning and deep learning, as evidenced in recent research, illustrate how these technologies are being harnessed to drive innovation, improve accuracy in decision-making, and foster development of more intelligent, self-learning systems within the software engineering realm [16]. Together, these studies underscore the pivotal role AI is playing in reshaping software engineering, marking a new era of sophistication and capability in architectural decision-making and the broader field.

1.1. Problem Statement and Research Questions

The architectural decision-making process is a critical aspect of software development, impacting the overall quality, maintainability, and evolution of the system. Despite its significance, this process often encounters various challenges, including the complexity of choices, the need for timely decision-making, and the documentation and justification of these decisions. Understanding how these decisions are made, the factors influencing

them, and the methods employed to document and justify them is essential for improving architectural practices and outcomes. We identified the following research questions to guide our study, as stated below.

- **RQ.1.** How are architectural decisions made in practice, how are these decisions documented, and how they affect each other?
  - **RQ.1.1.** Is there a certain approach to decision-making and how many people usually make decisions?
  - **RQ.1.2.** What information is kept for each architectural decision and where are the decision documents stored?
  - **RQ.1.3.** How to track decisions that affect each other?
- **RQ.2.** What factors usually influence architectural decisions and what are the compelling factors faced when making decisions?
  - **RQ.2.1.** Do perceptions regarding challenging situations in architectural decision-making change with respect to whether these architectural decisions are documented, the development lifecycle model used, and whether decisions are made individually or as a group?
  - **RQ.2.2.** Do perceptions regarding the factors that affect the architectural decisions change with respect to whether these architectural decisions are documented, the development lifecycle model used, and whether decisions are made individually or as a group?
- **RQ.3.** How is the choice made among architectural decision alternatives and how is the final decision justified?
- **RQ.4.** What improvements are needed to expedite the architectural decision-making process and make better architectural decisions?
- **RQ.5.** Can an AI-supported tool be developed to mitigate the challenges encountered in the architectural decision-making process?

- **RQ.5.1.** Would decision-makers adopt such a tool in their architectural decision-making workflow?

The details of the studies conducted to answer the research questions identified above and the answers found to these research questions are detailed in Sections 3. and 4.

1.2. Scope of the Thesis

With the advancement of AI, its application in the field of software development has become increasingly prominent. AI technologies are now frequently integrated into various aspects of software engineering [17]. In the context of software architecture, AI aids in navigating the complexities of decision-making. Traditionally, software architecture decisions involve considering various factors like system quality, e.g., performance, and maintainability. These decisions require balancing technical constraints, business objectives, quality attributes, and budgetary limitations. AI's role in this process is transformative, providing tools and methodologies to simplify and improve decision accuracy.

This thesis aims to design, develop, implement and validate a Software Architecture Decision Support System (SOFAR-DSS), an AI-driven tool to assist software architects in making informed and efficient decisions. The primary goal of this thesis is to develop a tool that significantly simplifies the architectural decision-making process in software engineering. This involves automating the analysis of complex decision matrices and providing intelligent, data-driven insights to reduce the cognitive burden on architects. The development of SOFAR-DSS involves the creation of advanced machine learning models, integration of XAI for transparency, and application of user experience design for ease of use.

To the best of our knowledge, there has been no prior research that specifically explores the use of a tool like SOFAR-DSS, which integrates artificial intelligence into the architectural decision-making processes within software engineering. This thesis, therefore, fills a significant gap in the existing literature by developing, implementing, and validating an AI-driven decision support system that addresses the complexities and nuances of software

architecture design process. Our research not only contributes a novel tool to the field but also provides valuable insights into the practical application of AI in enhancing the decision-making process in software engineering, particularly in architectural design.

1.3. Contributions

In this thesis, we introduce an innovative AI-based tool, SOFAR-DSS, designed to ease the decision-making process. SOFAR-DSS automates complex decision matrix analysis and provides intelligent insights, thereby reducing the cognitive load on architects and decision makers. This allows them to focus more on the design aspects of software development without being overwhelmed by technical choices. The system's recommendations are validated using the DBpedia ontology [18], [19].

SOFAR-DSS stands out for its ability to analyze architectural decisions within the context of specific projects. Unlike traditional methods, it dynamically adapts to each project's unique constraints and requirements, offering a more tailored and context-aware approach. It leverages advanced machine learning algorithms to process diverse data, including historical project outcomes, market trends, and technological developments. This capability enables the system to provide proactive predictions and recommendations.

Furthermore, SOFAR-DSS incorporates explainable artificial intelligence (XAI) [20], enhancing transparency in the decision-making process and allowing architects to understand and trust the system's logic. This cross-disciplinary approach, merging software engineering, data science, and cognitive computing, significantly improves architectural decision-making. The system's combination of AI-driven insights with an interactive framework empowers architects to make more informed, efficient, and strategic decisions.

In this thesis, we delve into the prevalent gaps in the architectural decision-making domain and introduce a pioneering approach that stands out for its simplicity, efficiency, and innovative edge. The primary contributions of our thesis are delineated below, each building on the premise of enhancing the current standards and practices in software architecture:

- **Extensive Analysis Through a Comprehensive Survey:** We embarked on an exhaustive exploration of the architectural decision-making landscape by administering an in-depth survey. This survey, unparalleled in its scope within the current literature, engaged 101 professionals, encompassing a wide array of expertise in the software architecture realm. The meticulously designed questions ensured a holistic examination of the decision-making process, covering every stage from the initial conceptualization to the finalization of architectural decisions. The richness of the data gathered provides unprecedented insights, offering a granular understanding of the intricacies involved in making architectural choices.

- **Identification of Key Challenges and Influencing Factors:** Our research meticulously examines every segment of the architectural decision-making journey, unlike previous studies that focused on just a single aspect, thereby conducting detailed investigations across all phases of the process. This comprehensive research has enabled us to identify the key challenges and factors that critically shape the trajectory of architectural decisions. The findings illuminate the multifaceted nature of decision making in software architecture and shed light on areas previously unexplored or inadequately addressed in existing academic studies.

- **Development of SOFAR-DSS:** Based on insights from our extensive research, we designed and developed the SOFAR-DSS tool, an innovative solution aimed at reducing the decision-making burden of architects in the architectural decision-making process. This tool enables the integration of artificial intelligence in the architectural decision-making domain, specifically by utilizing widely adopted question answering AI models to generate logical and relevant recommendations.

  By synergizing AI with the comprehensive DBpedia ontology, we have not only improved our recommendations but also achieved a form of validation that ensures they are both robust and relevant. This integration enables SOFAR-DSS to provide recommendations that are not only contextually aware but also deeply rooted in software architecture best practices, ensuring that it is well aligned with the complexities and evolving dynamics of contemporary software projects.

Furthermore, to emphasize the transparency of our system's inner workings and increase user trust, we have included explainable artificial intelligence (XAI) functionalities. This feature explains the logic behind AI-driven recommendations, giving users a clear understanding of why certain recommendations are made. This transparency is crucial to foster a sense of trustworthiness and enable users to make informed decisions based on AI insights.

The development of SOFAR-DSS represents a significant leap forward, combining the precision of AI with the strategic depth of architectural decision-making. This tool not only anticipates the needs of modern software projects, but also intelligently adapts to them, providing recommendations that are both actionable and insightful. With SOFAR-DSS, we provide a sophisticated, AI-enhanced tool that addresses the complexities and challenges inherent in the architectural decision-making environment.

The development and validation of SOFAR-DSS included a comprehensive process with experts from five different companies. We conducted a survey to evaluate the system's usability and areas for improvement, and the feedback was highly positive. Experts praised the system's user-friendliness and expressed their intention to recommend it for use in their projects. This expert validation confirms the system's practical applicability and potential impact in real-world scenarios.

- **Bridging AI and Decision-Making in Software Architecture:** SOFAR-DSS represents a significant stride in combining AI with decision-making processes in software engineering, addressing a notable deficiency in current academic discourse. The introduction of SOFAR-DSS marks a significant advance in the integration of AI into the strategic processes of software architecture. This integration will transform how decisions are made in software engineering and fill an important gap in how AI is used in this field. Our tool simplifies and improves the decision-making process and lays the groundwork for future innovations where AI can further enhance the capabilities of software engineering professionals.

Each of these contributions collectively underscores the novelty and the potential impact of our research, setting a new benchmark for future explorations and practical applications in the realm of software architecture and decision-making. Additionally, our work has resulted in two journal articles [21, 22], further disseminating our findings and providing a foundation for continued advancements in this field.

1.4. Research Design

This section describes the methodological framework that underpins our thesis and provides a comprehensive roadmap of the research followed throughout the thesis. The journey of this research began with the rigorous formulation of research questions designed to link the study to key research themes and guide the subsequent methodological steps. A visual representation summarizing the process from the first to the last step of this thesis and the outputs achieved at each stage is provided in Figure 1.1.

In the first phase of the research, a semi-structured questionnaire survey was conducted. This methodological choice allowed for an in-depth exploration of the different perspectives and experiences of professionals in the field, thus generating rich qualitative and quantitative data that informed the initial stages of the research. The insights gained from this semi-structured exploratory study led to the preparation and presentation of a paper to be presented at a national conference.

Building on the baseline information generated through the semi-structured questionnaire and an extensive review of relevant literature, the survey questions were refined and expanded. This resulted in a comprehensive online survey that successfully engaged 101 participants. The data collected from this extended survey provided deep insights into the research questions, resulting in a journal article articulating the key findings and their implications for the field.

Leveraging the knowledge gained from both the semi-structured interviews and the online survey, SOFAR-DSS, an Intelligent Decision Support System (IDSS) specifically designed to improve decision-making processes in software architecture, was designed and

developed. The development and outcomes of SOFAR-DSS are comprehensively detailed in a subsequent journal article. This publication marks a significant advancement in our research and contributes a novel tool to the software engineering domain.

The research design of this thesis is therefore characterized by a phased approach, from initial question formulation to data collection and development of a practical tool. Each stage of the research was carefully planned to build on the previous one, thus ensuring a cumulative knowledge building process. The details of the survey study and the developed DSS are given in sections 3. and 4., respectively.

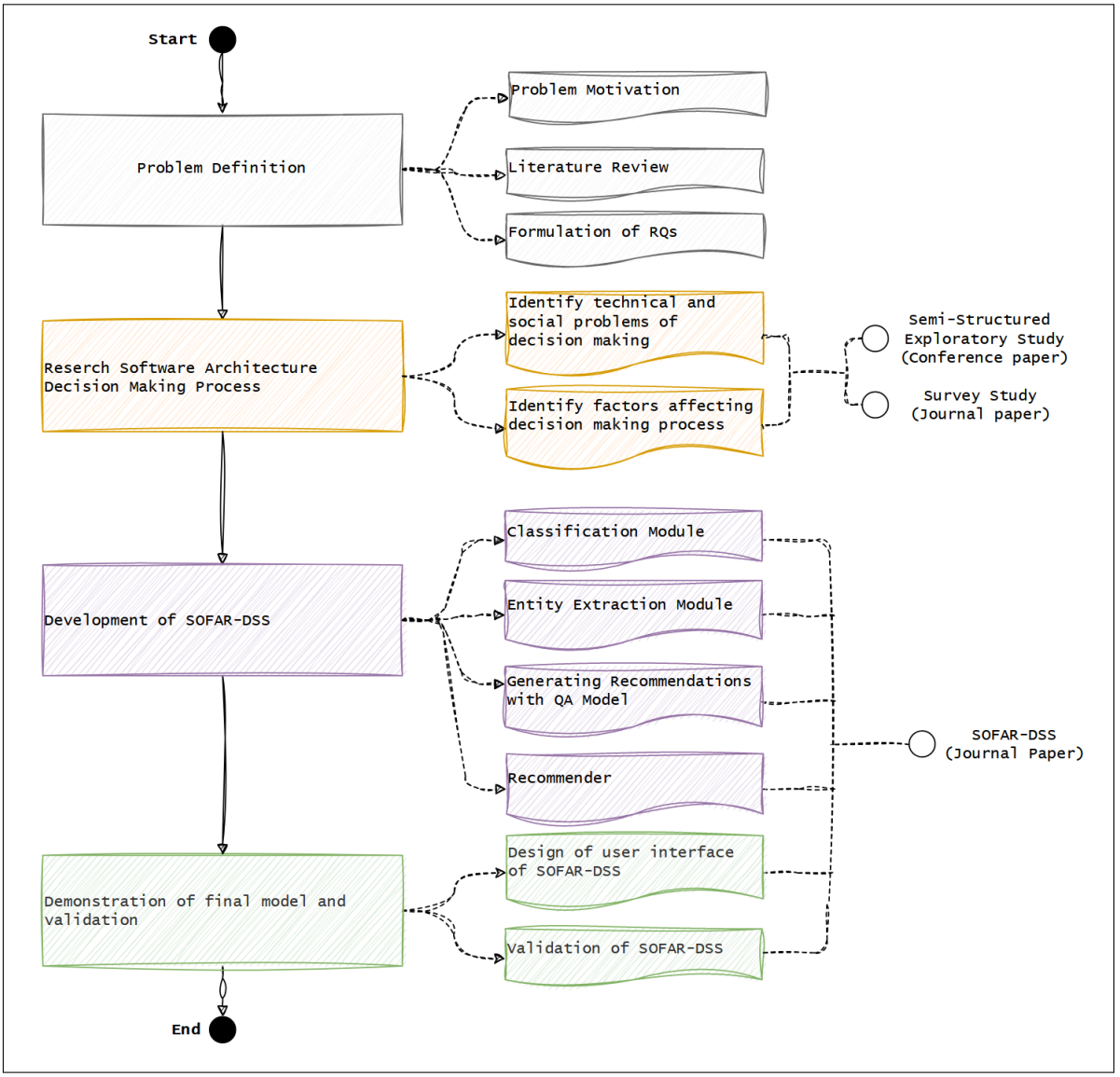


Figure 1.1 Graphical representation of the thesis roadmap

1.5. Organization

The organization of the thesis is as follows:

- Chapter 1 gives a brief introduction to the problem, presents our motivation, the scope of the thesis, contributions, and methodological framework.
- Chapter 2 provides background on relevant topics, including softare architecture, design patterns.
- Chapter 3 describes the methodological framework and provides a comprehensive roadmap of the research followed.
- Chapter 4 details the survey study conducted to identify the challenges faced in the architectural decision-making process and the factors that influence this process.
- Chapter 5 introduces the intelligent decision support system that will facilitate the architectural decision making process.
- Chapter 6 states the summary of the thesis and possible future directions.
- Chapter 7 gives the results of the studies carried out.

2. BACKGROUND OVERVIEW

2.1. Software Architecture

Software architecture plays an indispensable role in the realm of software engineering, serving as the cornerstone for crafting efficient and effective software systems. With the increasing size and complexity of these systems, the imperative to manage this complexity and ensure the system's integrity becomes paramount. The adoption of a well-defined architectural framework is not merely beneficial but essential for the successful navigation and guidance of a project towards its completion [3]. This structured approach empowers software engineers to adeptly manage the intricacies inherent in the structures they devise, facilitating a more manageable development process.

The concept of "software architecture" lacks a single, universally accepted definition, which is reflective of its broad applicability and the diversity of thought within the field. Instead, it has been the subject of various interpretations by scholars and practitioners alike, each bringing their perspective to bear on the understanding of what constitutes software architecture [23]. Among the many definitions, the one proposed by Bass et al. stands out for its clarity and wide acceptance within the community. According to them, "the software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them. Architecture is concerned with the public side of interfaces; private details of elements—details having to do solely with internal implementation—are not architectural."[3]. This definition underscores the multifaceted nature of software architecture, highlighting its role in facilitating the conceptualization and visualization of a system's organization, behavior, and key attributes.

Furthermore, software architecture acts as a blueprint for both the system and the project, guiding the development process and providing a frame of reference for understanding the system's organization and expected behavior. It assists in identifying potential design issues early in the development lifecycle, thereby mitigating risks and reducing the

likelihood of costly rework. Moreover, it supports scalability, performance optimization, and the achievement of other quality attributes by defining clear interaction patterns among components and specifying the architectural styles and patterns to be employed.

In essence, software architecture embodies the strategic choices that shape the technical foundation of a software system. These choices have far-reaching implications for a project's overall direction, affecting aspects such as system scalability, maintainability, and the ability to meet user requirements and adapt to changing technologies. As such, the development of a software architecture requires careful consideration, foresight, and a deep understanding of both the problem domain and the technological landscape. By laying the groundwork for all subsequent design and implementation decisions, software architecture remains a critical, guiding force in the creation of successful software systems.

Expanding further, the concept of software architecture can also be viewed through the lens of architectural styles or patterns, each offering a different method for structuring software systems. Notable examples include the layered architecture, event-driven architecture, microservices architecture, and service-oriented architecture, among others. These styles provide templates that help in addressing specific system requirements and often embody best practices that have evolved within the field of software engineering [24]. Shaw and Garlan's definition emphasizes architecture as the highest level of abstraction in a system, beyond the algorithms and data structures of the computation design. They argue that architecture is about the overall structure of software and the principled way of designing the system, which includes choices about the organization, security, performance, and reliability of the software application [24].

Kruchten further elaborates on the practicality of software architecture, introducing the concept of the "4+1" view model of architecture. This model provides a framework for describing the architecture of software-intensive systems, based on the use of multiple, concurrent views. The views address separate concerns of various stakeholders of the system and include scenarios to illustrate how the architecture responds to external stimuli [25].

Additionally, Clements et al. contribute to the understanding by asserting that software architecture necessitates a series of decisions based on quality attributes. These decisions ultimately influence the system's non-functional characteristics, such as security, modifiability, and performance, ensuring that the architectural framework aligns with both functional and non-functional requirements [26].

Richards et al. define software architecture in four dimensions: the structure of the system (refers to the type of architecture style(s) such as microservices, layered, or microkernel), combined with architecture characteristics ("-ilities", success criteria of a system) the system must support, architecture decisions (define the rules for how a system should be constructed), and finally, design principles (differs from an architecture decision in that a design principle is a guideline rather than a hard-and-fast rule).

Incorporating this comprehensive perspective enriches the understanding of software architecture's role in system development. The four dimensions outlined by Richards et al. offer a holistic view of what software architecture entails, providing a multifaceted framework that supports the systematic development of software systems. This framework not only guides the architectural design but also integrates it with the operational and strategic goals of the system, ensuring that the architectural solutions devised are robust, scalable, and in alignment with the stakeholders' expectations. By considering these dimensions, architects can create systems that are not only technically sound but also adaptable to changing requirements and capable of delivering sustained value over time [27].

Thus, the inclusion of these four dimensions into the conceptualization of software architecture allows for a richer, more nuanced understanding of its importance. It highlights how a well-thought-out architectural framework can influence the success of a system, facilitating the achievement of both functional and quality requirements. This multi-dimensional approach underscores the complexity and strategic significance of software architecture, illustrating how it underpins the entire software development process and contributes to the realization of a coherent, well-functioning software system.

In summary, software architecture remains a crucial element in software engineering, providing a strategic framework that shapes the development and integrity of software systems, especially as they increase in size and complexity. It encompasses a set of fundamental structures (components, their interactions and properties) that guide the design of the system and underlie the conceptualization of its organization and behavior. While its definitions vary to reflect the breadth and depth of the field, common themes include its role as a blueprint that determines the structure, behavior and quality attributes of the system and facilitates early identification of potential design challenges. Definitions by Bass et al. and Richards et al. emphasize its multifaceted nature, including elements such as architectural styles, decision-making criteria and guiding principles. This holistic perspective underscores the critical role of software architecture in ensuring that software systems are efficient, reliable and capable of meeting evolving technological and user requirements, thus confirming its fundamental importance in the successful engineering of complex software solutions.

2.2. Software Architectural Decision Making

Making decisions in software architecture involves considering a multitude of factors, with each decision having a significant impact on the software's quality, performance, maintainability, and overall success [1]. The architecture of a system is shaped by numerous decisions [5], and this decision-making process is fraught with uncertainties, such as conflicting stakeholder objectives, unpredictable project estimates, uncertain availability of resources, and the unknown effects of decisions on stakeholder goals, making it a complex and challenging task [28]. Traditionally, decision-making in this context is about making choices that significantly influence the entire software system [6]. This process entails identifying various alternatives and selecting one as the final decision. Understanding these alternatives is crucial for effective decision-making, which in turn directly influences the quality of the system [29].

Software architecture also serves as a vital communication tool, articulating the design decisions and their outcomes to stakeholders [30]. However, it's important to recognize that the knowledge and information about these design decisions are often implicitly embedded within the architecture, making it challenging to explicitly extract them, which can lead to loss of knowledge [26]. Architectural documentation is essential in facilitating communication among stakeholders, especially when there are multiple people on a development team, when the implementer and the architect are different individuals, or when the team is geographically dispersed [26, 31, 32]. Such documentation, which includes architectural decisions and their justifications, system components and their relationships, and chosen design patterns, significantly impacts the development and maintenance phases [33]. Jansen et al. [31] have noted several benefits of architectural documentation, such as enabling asynchronous communication among stakeholders, mitigating the loss of architectural knowledge [34], constraining implementation, shaping organizational structure, promoting reuse of architectural knowledge, and aiding in training new project members. Architectural documentation sets out the rules for achieving correct system behavior [35] and is a key factor in sharing and reusing architectural knowledge [31]. Recently, the importance of thoroughly documenting architectural decisions in software projects has been increasingly emphasized [36–38]. However, this documentation is often neglected, as Jansen et al. [34] noted, with architectural knowledge being documented "after the fact." This is partly because documenting architectural design decisions is time-consuming and effort-intensive [39], and many developers are unsure about what to document, leading to only a few teams taking the time to record their decisions [26, 40, 41].

2.3. Design Patterns in Software Architecture

Design patterns in software architecture represent a set of best practices that have been developed and refined to address recurrent design problems in software development. They provide a template or blueprint that can be applied to a design problem within a given context, facilitating the creation of software architectures that are both robust and maintainable [42]. Design patterns encapsulate the expertise of seasoned software developers and architects,

offering proven solutions that can expedite the development process and enhance the quality of software systems.

Incorporating design patterns into the architectural decision-making process is beneficial for several reasons. First, they promote design reuse, which can lead to a more efficient development cycle by reducing the time and cost associated with the invention of new solutions [35]. Second, patterns can increase the understandability of the architecture by providing a common language for developers and stakeholders [26]. This shared vocabulary facilitates clearer communication and helps in aligning the team's understanding of the system's design [43]. Furthermore, design patterns can also contribute to the system's scalability and flexibility by ensuring that the architecture is composed of well-defined, interchangeable components [44].

Each design pattern typically addresses a specific aspect of software architecture, such as structural organization, communication between objects, or the behavior encapsulation within a system [45]. Patterns are often categorized into three main types: creational, structural, and behavioral patterns [42]. Creational patterns deal with object creation mechanisms, structural patterns address the composition of classes or objects, and behavioral patterns characterize the ways in which classes or objects interact and distribute responsibility [46].

Applying design patterns in the early stages of architectural design can prevent issues that might arise later in the development lifecycle. It can also assist in identifying potential risks and trade-offs associated with each pattern, allowing architects to make more informed decisions [6]. In practice, the selection and implementation of design patterns must be done judiciously, as each pattern comes with its own set of consequences and implications for system performance, maintainability, and future adaptability [47]. As such, a thorough understanding of both the problem space and the available design patterns is critical for architects aiming to leverage these constructs effectively within their software architecture.

2.4. Decision Support Systems

Decision Support Systems (DSS) are acknowledged as computer-aided tools intended to support administrative decision-makers in addressing non-structural problems [48]. Over the past four decades, DSS have undergone a significant evolution, transitioning from theoretical frameworks to practical, computerized applications in various fields, including software engineering. The architecture of a modern DSS is fundamentally built on three core components: a knowledge base, a computerized model, and a user interface [49]. The knowledge base serves as a repository of accumulated information and data, crucial for informed decision-making processes. This component is especially relevant in software engineering, where evolving best practices and historical data play a key role in project planning and execution. The computerized model, another critical element of DSS architecture, facilitates the analysis and simulation of various scenarios. In software engineering, these models can represent systems in development, helping in predicting outcomes of different methodologies or strategies before their actual implementation. They are particularly useful in choosing appropriate Software Development Life Cycle (SDLC) models and managing risks and changes in software projects. Lastly, the user interface in a DSS is designed to be intuitive, enabling easy navigation and manipulation of data.

Integrating AI methodologies like expert systems, data mining, and machine learning, DSS simulate the cognitive decision-making functions of humans [50]. This allows for advanced decision support functions, enhancing the efficiency and effectiveness of decision-making in software engineering. IDSS, a subset of DSS, utilize AI to analyze large data sets, identifying trends and patterns that aid in making more informed decisions. Thus, the evolution of DSS over the years has led to the development of sophisticated tools that significantly aid in decision-making processes in software engineering, aligning with the industry's need for data-driven and intelligent solutions.

3. FACTORS AFFECTING ARCHITECTURAL DECISION-MAKING PROCESS AND CHALLENGES IN SOFTWARE PROJECTS

Decision making in software architecture goes beyond the simple application of predefined methods; it involves human participants, each with their own unique approach to decision making [51]. To understand the practical aspects of this process, we carried out a survey to investigate the methods of decision-making, how these decisions are recorded, the storage and management of decision documents, the technical and social obstacles encountered, areas needing improvement, and the requirements for better decision-making. This approach is akin to Weinreich et al.'s [52] expert survey, which also examined the types, influencing factors, and documentation of design decisions in the field. Our goal is to pinpoint changes and influences in architectural decision-making and propose ways to make documentation more organized and effective. Our study differs from Weinreich et al.'s [52] in that we delve deeper into every phase of the architectural decision-making process, looking at the entire process from decision-making to documentation and verification. Like all organizational decision-making processes, architectural decision-making involves preceding and subsequent activities that impact and are influenced by the decisions. Previous studies (e.g., [40, 52, 53]) have concentrated on specific aspects of the decision-making process, often neglecting the preceding and following activities. Our study seeks to bridge this gap by systematically examining each stage of the decision-making process, including previously unexplored phases like identifying architectural requirements and verifying decisions. We believe our study is the first to comprehensively survey and understand the architectural decision-making process in its entirety, considering the influencing factors, challenges, and the subsequent implementation and documentation. Furthermore, our study aims to uncover potential research areas illuminated by our findings. Ultimately, our motivation is to generate new, actionable insights based on the results of our research.

3.1. Related Work

In the field of literature, numerous studies have focused on understanding the methodologies employed in the decision-making process for software architecture and design. These studies have explored the nature of decisions made, their documentation methods, influencing factors, and the challenges faced during the process.

Dasanayake et al. [40] conducted a case study to explore the decision-making approaches of professional software architects in the industry. Their objective was to identify the various approaches to architectural decision-making, the reasoning behind these approaches, associated challenges, and potential enhancements for better decision-making in architecture. Their findings revealed insights into the decision-making practices in software architecture across three small to medium-sized enterprises. The study found that software architects often rely on informal, yet structured methods at the team level, while individual decision-making is largely influenced by personal traits like intuition and experience. Factors such as budget, time constraints, and organizational practices were identified as hindrances to comprehensive analysis prior to decision-making.

Miesbauer and Weinreich [54] carried out an expert survey to investigate the types, influencing factors, and sources of design decisions, along with their documentation in practice. The survey, involving software architects, team leaders, and senior developers from six Austrian companies with an average of over 10 years in software development, aimed to understand decision-making and documentation processes in practice, types of decisions documented, and the driving forces behind these decisions. The study highlighted that architects and designers mainly focus on technological and structural decisions influenced by constraints and user requirements, and these decisions are typically well documented.

Building on their 2013 work [54], Weinreich et al. [52] conducted another expert survey to delve deeper into the types, influencing factors, and sources for software design decisions, and their practical documentation. This survey included 25 participants from 22 companies across 10 countries, all with over 13 years of experience in software development. The

findings validated their earlier work on design decision classification and influencing factors, while also uncovering additional decision types and influences not previously identified. The study provided insights into the relative importance of various decisions and their influences, and potential sources for decision recovery, revealing a preference against formalized documentation methods due to time and cost constraints.

Tofan et al. [4] analyzed the characteristics and contributing factors of 86 real-world architectural decisions based on a survey of 43 industry architects. The study compared decisions made by junior and senior architects, revealing that architectural decisions typically take eight working days, with junior architects spending a quarter of their time on decision-making. Good architectural decisions were found to involve more alternatives than bad ones, and 86% of these decisions were made collaboratively.

Muccini et al. [53] examined industrial practices in group decision-making for software architecture with 35 global participants. Using both qualitative and quantitative methods, they discovered a preference for unstructured, discussion-based decision-making approaches in software architecture groups, differing from those documented in literature due to budget, time, and resource constraints, leading to various group conflicts. The creation of prototypes was identified as a crucial step in group decision-making.

Hofmeister et al. [55] highlighted three key architectural activities where critical decisions are made: architectural analysis, synthesis, and evaluation. Architectural analysis involves filtering concerns and context to establish architecturally significant requirements, synthesis proposes solutions to these requirements, and evaluation ensures the appropriateness of the decisions made.

These studies collectively examined Hofmeister et al.'s [55] three decision-making activities from various angles, including decision-making practices [40], challenges encountered [4], decision classification [54], factors affecting and documenting decisions [52], and group decision-making [53]. Notably, most studies focus on specific aspects of the decision-making process or have a limited participant pool. Our study stands out by engaging a larger and more diverse group of software architecture experts, offering a detailed evaluation of the

decision-making process from various perspectives. We conducted an in-depth analysis of each stage of the decision-making process. We aimed to understand the content, accessibility, and traceability of decision documents in practice to lay the groundwork for future research aimed at systematizing decision documentation.

3.2. Methodology

We conducted two studies to identify the factors affecting the architectural decision-making process and the challenges encountered in this process. The first was a semi-structured exploratory study [21] with nine experts to delve into their practical experiences in architectural decision-making and the overall process, and the other was an online study [56] with 101 experts. Ethical approval has been obtained from the Hacettepe University Ethics Committee for the conducted both semi-structured exploratory study and survey study. Ethical approvals for both studies can be found in Appendix C (Figure 6.1) and Appendix D (Figure 6.2).

3.2.1. Semi-structured Exploratory Study

This study was conducted as an exploratory qualitative research. Qualitative research is suitable for the objectives of our study as it focuses on examining subjects in their natural settings, exploring the reasons indicated by respondents, and understanding their perspectives regarding the issue [57–60]. It employs qualitative data collection methods such as observation, interviews, and document analysis to perceive solutions to a problem, allowing for the recognition of previously known or unnoticed issues and addressing the natural phenomena related to the problem through a subjective-interpretative process [61]. This method aims not only to understand what people think but also why they think that way. It involves considering subjective data such as individuals' opinions, experiences, perceptions, and emotions. The data collection method used in this study involves semi-structured interviews with open-ended questions. The interview questions were structured around four main topics: general information about the expert, details

related to architectural decision-making, techniques and tools used in decision-making, and expectations regarding the improvement of the decision-making process. During the interviews, questions related to these planned topics were advanced in a flexible manner, adapting to the flow of the conversation without strict adherence to a predetermined order, facilitating a dynamic question-and-answer exchange.

This exploratory study provided detailed insights into the practical process of architectural decision-making among architects and developers, interpreting the results obtained from expert consultations. The recordings made during the interviews were analyzed, categorized according to the questions, and key contents were identified. The analyzed results highlighted how architectural decisions are made, the influential factors during decision-making, and the technical and social challenges encountered in this process. The findings of this study aim to identify different software architectural decision-making practices followed by software teams, determine the influential factors and associated challenges, and suggest potential improvements for making sound architectural decisions from the perspective of software architects.

The objective of this research is to gain a comprehensive understanding of how each stage of the architectural decision-making process is conducted in practice, including making, documenting, and operationalizing decisions, identifying factors influencing the decision-making task, and pinpointing the technical and social obstacles that complicate decision-making. As part of this goal, we primarily sought to learn how architectural decisions are realistically made and documented.

3.2.1.1. Population

In this study, we targeted software engineers, team leaders, and software architects from various companies in Turkey, specifically those with more than three years of professional experience in software architecture and development. Consequently, we conducted interviews with nine experts from nine different companies, all of whom possess expertise in industrial and corporate software system development. Among the participants, eight

were male and one was female. These individuals have gained experience in more than three companies in the field of software development and have an average of ten years of experience, with a minimum of three years in each of these companies. The demographic characteristics of the interviewees are presented in Table 3.1 below. Table 3.2 provides detailed information for each participant, including gender, title, total years of experience, number of companies worked for, number of different titles held, the number of software engineers/developers in their current company, the number of employees in the current project, the number of developers in the largest project they have been involved in date, the city they are currently working in, and the duration of the interview. In this study, participants responded to all questions considering their past experiences, which also influence their responses, especially the size of the projects they have been involved in the past. According to the information in Table 3.2, considering the projects the participants have been involved in date, the number of developers in the largest project they have worked on ranges from 10 to over 100.

Table 3.1 Participant Demographic Information

Gender	**Count**
Female	1
Male	8
Title	
Software Architect	5
Team Leader	4
Experience	
0-9 years	1
10-19 years	5
20 years and above	3

Table 3.2 General Information About Participants (P1-P9)

Participants	Gender	Current Title	Total Experience (Years)	Number of Companies	Different Titles Held	Number of Engineers/Developers	Current Project Size	Largest Project Size	Current City	Interview Duration
P1	Male	Software Architect	27	7	6	20	20	20	Ankara	90min
P2	Male	Software Architect	25	6	6	240	10	45	Ankara	45min
P3	Male	Team Leader	14	7	4	500	10	50	Istanbul	41min
P4	Male	Team Leader	20	4	5	2	3	12	Ankara	77min
P5	Male	Team Leader	12	6	4	20	12	100+	Ankara	45min
P6	Female	Software Architect	7	2	4	50	34	50	Istanbul	60min
P7	Male	Software Architect	17	2	4	40	12	50	Ankara	67min
P8	Male	Software Architect	14	6	4	180	18	10	Ankara	40min
P9	Male	Team Leader	18	5	4	12	16	70	Ankara	50min

3.2.1.2. Data Collection

During the interviews, open-ended questions were asked to gather information about the experts' professional experience, previous projects and teams, and the approaches they use in architectural decision-making. At the same time, it was ensured that they could answer more comfortably and sincerely by asking the questions in an impromptu manner without observing a specific order for the questions. The questions were addressed under four different topics. These are: questions about the expert, questions about the decision-making process, questions about the techniques and tools used in decision-making, and questions about experts' expectations about their decision-making approach. The first part (two

questions) aims to learn more about the experts and their background. The second part (eight questions) aims to identify the decision-making and documentation process, the factors influencing this process and the technical/social problems encountered in this process. It aims to collect examples of architectural decisions and architectural decision categories. The third section (six questions) focuses on the techniques and tools used in the decision-making process. The last part (one question) aims to find out the expectations for the improvement of the areas discussed during the interview. The interviews with the participants were conducted online and each expert interview lasted on average of 57 minutes. Prior to the interview, the participants' permission was obtained and the interviews were recorded. The questions asked to the participants are given in Table 3.8 as source [21].

3.2.1.3. Data Analysis

For the confidentiality of participants' personal information, each participant was assigned a different code (e.g. P1). In order to prepare all interview recordings for analysis, the recordings were transferred to a fixed form and transcribed, and then the transcripts were grouped by question. According to this grouping and data content, themes called "key content" were created. The frequency and percentage values of these key contents were calculated according to the person and question-based groupings.

3.2.1.4. Findings

This qualitative study was carried out with participants from nine different companies in Turkey, targeting software engineers, team leaders, and specifically those with over three years of professional experience in software architecture and development. As a result, interviews were conducted with nine experts—five software architects and four software team leaders—each with a minimum of ten years of professional experience, barring one participant. During our study, we gathered information on how decisions are currently made and captured in practice, the types of decisions made and documented, where

they are stored, the techniques and tools used in the decision-making process, and the factors influencing architectural decisions. Our research was organized under four different headings: general information about the expert, decision-making, techniques, and tools used in decision-making, and expectations for improving the decision-making process.

Questions About the Expert

At the beginning of the interviews, information was obtained about the participants' previous experiences, the roles they were assigned, and their years of experience. The research has been conducted with participants having various years of experience, as shown in Table 3.1. The total years of experience among the participants are at least 7 years. All participants have held roles as software architects or team leaders for at least 2 years. In their past experiences, they have worked in various roles such as research associate, team leader, software architect, front-end developer, manager, and consultant. The projects they were involved in, in terms of the size of the developer team, ranged from 5 to 150 developers.

Questions Related to Decision-Making

In this category, questions were asked specifically for RQ1 and RQ2 to gather information. Participants were asked various questions regarding decision-making within the scope of the study. To gain insights into how architectural decisions in the software architecture decision-making process are made, who makes these decisions, which decisions are documented, where the documented decisions are stored, the factors influencing decision-making, and the technical and social difficulties that complicate the decision-making process, questions were posed to the participants in this context. The questions asked in this scope and the participants' responses are provided below.

What do you think software architecture is? Participants provided various answers to the definition of software architecture. However, at the core of all responses was the definition of software architecture presented by Bass et al. [3], which states: "The software architecture of a system is the set of structures needed to reason about the system, which comprises software elements, relations among them, and properties of both" [3]. Essentially, all participants

defined the software architecture of a program or a system as the structure or structures consisting of software components designed to meet specific needs and the relationships between these components. Some of the responses provided by the participants are as follows:

- *"A set of high-cost decisions" (P1)*
- *"Software architecture is a design, but not every design is a software architecture. It is the design you create to be able to construct a system and to continue maintaining it easily and efficiently, to be expandable. It is the design that clearly outlines the components that make up the system and the relationships between these components." (P4)*
- *"The backbone that the software can operate on and how the parts on that backbone can understandably communicate with each other." (P7)*

Which decisions are documented? Five participants mentioned that they document all decisions, two stated that the documentation of decisions is partial, and only two participants indicated that they do not formally document decisions but rather evaluate and experiment with suitable alternatives according to the project's requirements and specifications. Some of the responses from the participants are as follows:

- *"We do not document all architectural decisions, only some are documented as evidence." (P4)*
- *"Decisions are made within the scope of the technical specification. All derived designs are documented and recorded." (P6)*

When are these documented decisions recorded, and where are they stored? Most participants noted that the decision-making process is more intense in the initial stages of the project life cycle, with subsequent stages primarily involving modifications to the decisions already made. Some responses regarding where the documented decisions are stored are as follows, with other participants providing largely similar responses:

- *"I create and store them in collaborative environments like Confluence or Google Drive." (P1)*
- *"We have implemented Architectural Decision Records (ADR) documents. When a team chooses something, it states why it chose it, the implications of this choice, and the pros and cons in the ADR documents." (P3)*

The responses given by the participants to questions related to the decisions made are presented in Table 3.3. According to the data in the table, there is either full or partial access by all developers to the decisions made, and there are no decisions that have not been consciously documented. These decisions are made through consultation by competent individuals coming together, depending on the company hierarchy. Generally, if the projects are similar, the participants' responses were inclined towards using previously made and successful decisions in different projects. Many participants indicated that the content of the project and the success of the decision are decisive in such a scenario.

Table 3.3 Responses from participants regarding other questions related to the decisions made.

Question	**Answers**	**Count**
1. Can all developers access these decisions?	Yes	6
	No	0
	Partially	3
2. Are there any decisions that are deliberately not documented in your projects or company?	Yes	0
	No	9
3. Who usually makes the decisions?	Only one person	0
	By a team	9
4. Do you reuse decisions made in various projects?	Yes	9
	No	0

What types of architectural decisions are typically made in practice? In response to this question, participants were presented with certain categories based on the classification

in the study by Weinreich and colleagues [54], and were asked to think about the decisions they encounter in practice and to indicate which category these decisions belonged to. All participants noted that they normally do not use a specific classification when making decisions in practice; however, they stated that they would mark according to the classification they considered upon reflection of their past decisions during the interview. One participant (P3) did not answer this question because they do not employ any classification in practice. These responses are shown in Table 3.4.

Table 3.4 Responses to the question about the types of decisions encountered in practice

What type of decisions do you encounter?	**Participants**								
	P1	**P2**	**P3**	**P4**	**P5**	**P6**	**P7**	**P8**	**P9**
Structural and behavioral decisions	x	x		x	x	x	x	x	x
Tools, technology, process, organization	x	x			x	x	x	x	x
Guidelines, design rules, desires	x	x		x		x		x	

What factors usually influence architectural decisions? Participants were asked to rate the factors that influence their decision-making process during the architectural decision-making stage on a scale of 5 (very influential), 4 (influential), 3 (neutral), 2 (not so much influential),), 1 (not influential at all). This question was also inspired by the study by Miesbauer et al. [54]. Accordingly, participants evaluated the factors presented to them based on our given scale. As seen in Table 3.5, the responses generally ranged between neutral and very influential. In the responses to the factors affecting architectural decisions, only P3 marked the value 1 (not influential at all). When considering the mode and median values, the factors that most influence architectural decisions were most often rated 4 and 5 by the users. As seen in Table 3.5, while user requirements were selected as the factor that most influenced architectural decisions, management preferences were chosen as the least influencing factor.

Table 3.5 Responses related to factors influencing architectural decisions

#	Factors Influencing Architectural Decisions	P1	P2	P3	P4	P5	P6	P7	P8	P9	Average	Mode	Median
1	Management Preferences	3	4	2	3	4	3	4	4	4	3.5	4	4
2	Tool and Technology Availability	3	5	4	2	5	5	4	4	4	4.1	5	4
3	Risk	5	4	5	5	4	4	5	5	5	4.5	5	5
4	Time	4	4	3	4	4	5	5	5	5	4.2	4	4
5	Quality Attributes	5	5	4	4	4	4	4	3	3	4.1	4	4
6	User Requirements	4	5	5	4	5	5	4	4	5	4.6	5	5
7	Maintenance / Product Life Cycle	5	4	3	4	4	4	4	3	3	4.0	4	4
8	Personal Experience / Preferences	4	5	1	3	3	4	4	4	4	3.6	4	4
9	Business Goals and Strategies	5	5	3	2	3	4	4	4	4	3.8	5	4
10	Previous Successful / Unsuccessful Decisions	4	5	5	5	3	4	4	3	4	4.2	5	4
11	Restrictions	5	4	1	3	4	4	3	4	4	3.6	4	4
12	Cost	5	5	1	4	5	5	4	5	5	4.3	5	5

Note: In this table, each factor is rated on a scale from 1 to 5, where 1 represents ”not influential at all” and 5 represents ”very influential”.

What are the factors that influence your architectural decision in terms of quality characteristics? Can you name the three most important quality factors? One of the participants (P6) did not answer this question and one participant (P5) listed only two quality factors. According to the answers of the other participants, scalability and performance are the most important quality factors when making architectural decisions. Testability, modularity, understandability, and maintainability were mentioned by only one participant each. The answers and frequencies of the participants are given in Table 3.6.

Table 3.6 Responses related to the factors influencing architectural decisions in terms of quality attributes

Quality Factors	**Participants**									**Frequency**
	P1	**P2**	**P3**	**P4**	**P5**	**P6**	**P7**	**P8**	P9	
Scalability	x		x	x					x	4
Performance	x	x			x		x			4
Extensibility				x			x	x		3
Reliability	x	x								2
Robustness			x				x			2
Safety		x	x							2
Availability				x					x	2
Testability								x		1
Modularity								x		1
Understandability									x	1
Maintainability					x					1

In the light of your experience in making architectural decisions in past projects, could you evaluate the influence of the following difficulties as 5 (very influential), 4 (influential), 3 (neutral), 2 (not so much influential), 1 (not influential at all), NE (Never Experiences)? What are your decision-making challenges? Thinking about the decisions you have made in previous projects, could you rate the following examples on a scale of 1-5. (1) strongly disagree, ... , (5) strongly agree. Participants were given the possible difficulties they faced in their past projects as shown as Q9 of Table 3.8 in Section 3.2.2.1.. Each participant evaluated the situations they faced in their past projects in terms of challenges considering the situations given as Q9 of Table 3.8 in Section 3.2.2.1.. The participants were asked to compare the given situations with the challenges they had faced in the past and if the given situation was very challenging, they were asked to strongly agree. agree (5), and not at all if the situation did not cause any difficulties in the past. We asked them to mark the option of disagree (1). As a result of the evaluations, F22 was selected as the most challenging

situation for making architectural decisions. F1 and F9 were selected as the least challenging situation.

What is Your Typical Process for Making Architectural Decisions? For this question, the participants stated that they do not use a specific approach and technique for the decision-making process. In general terms, considering the answers given to this question, the decision-making process takes place as follows: Get together, consult, identify alternatives, leave to work if needed, then get together again and make a decision.

Questions on Techniques and Tools Used in Decision Making

In this section, different questions were asked to the participants regarding the techniques and tools used in the decision-making process. According to the answers given by the participants, there were similar answers from the participants in determining the architecturally important requirements. Generally, important requirements are determined with what-if questions. When asked whether you look for alternative solutions to your needs even if you already have a solution in mind when making decisions, all participants, except for three participants, stated that they always look for alternative solutions. Three participants stated that they make a choice based on their past experiences and therefore do not always look for alternatives. Although there were different answers to the question of how do you choose between alternative solutions, basically most participants stated that the final decision is made by a team of experienced people. When asked whether they verify their final solution, some said that they use the ATAM method and do this when the system is commissioned, while others said that they do not use any verification method.

Questions on Prospects for Improving the Decision Making Process

In this section, participants were asked about areas that could be improved in order to make better decisions. A few different areas are given in the question as an example. Only one to participant (P7) did not answer this question. Effective information sharing among other participants except for the participants who did not select the field, all participants marked

the first field effective information sharing. As can be seen from Table 3.7, the frequency is the highest area is effective information sharing.

Table 3.7 Responses to the question about expectations for improving architectural decision-making

Which of the following areas could be enhanced to make better architectural decisions?	**Participants**									**Frequency**
	P1	**P2**	**P3**	**P4**	**P5**	**P6**	**P7**	**P8**	**P9**	
Effective information sharing	x		x	x	x	x		x	x	7
Keeping track of decisions and rationale	x	x		x	x	x			x	6
Improved documentation	x	x			x	x		x		5
Lightweight technique or tool to guide them	x	x						x		3
Making decision-making more agile			x			x			x	3

To summarize, the findings of our study indicate that although the software architecture decision-making process varies among companies, the underlying methodology is fundamentally similar. No specific approach or technique is consistently used for decision-making. It has been found that many companies tend to document the decisions made in the decision-making process. However, it has been expressed by the participants that stating the reasons explicitly in the documentation process is not preferred, as detailed documentation is often avoided by developers for being burdensome, and documenting simple and significant points would be sufficient. According to the responses given by the participants, the main factors affecting architectural decisions are user requirements, risk, and cost. However, participants have also emphasized the importance of personal experience and preferences. In terms of quality attributes, scalability, performance, and extensibility

have been seen as factors affecting architectural decisions. Participants have stated that the impact of these quality factors varies from project to project. For instance, scalability is not very effective for small projects but is significant for large-scale projects. They have also mentioned the technical and social challenges encountered in the decision-making process. The coercive effects of situations faced in each project differ, but the need for information to resolve uncertainty is usually a challenging situation in decision-making. Other challenging situations include the insufficiency of time to make decisions. The lack of alternatives for a decision and receiving conflicting recommendations from different sources have not been seen as having a challenging effect on the decision-making process. According to the responses given to the questions about the techniques and tools used in decision-making, the majority of the participants in the study have not validated their final solutions with methods like ATAM or CBAM, applying similar approaches when selecting among alternatives. The responses to improving the decision-making process are also of similar nature. All participants except one have stated that effective information sharing would contribute greatly to improving the decision-making process and leading to better decisions, along with following the design decisions and logic. In general, according to the results of the study, how the software architecture decision-making process is carried out, the factors influencing this process, the importance of documentation for architectural decisions, important areas for improving the architectural decision process, and the expectations of developers and architects have been revealed. During the interviews with the participants, they provided feedback considering the companies they have worked with so far. Based on this feedback and the answers to the survey questions, we can generalize that there is no systematic way for the software architecture decision-making process, documentation is mostly done more detailed and properly in military projects, and no method is used to verify the decision made.

3.2.2. Survey Study

Following the research questions outlined in Section 1.1., an questionnaire was crafted and disseminated online. In the development of the survey, we incorporated findings from

our preliminary study on the subject. Additionally, we referenced multiple studies for crafting questions related to decision-making influences, decision types [52], challenges within the decision-making framework [4], decision-making strategies, and the identification of stakeholders in the decision process [53]. The sources for the formulated questions, derived from various references, are noted in the reference section of Table 3.8. These questions were either adopted as is or slightly modified. The analysis of the survey responses was conducted on a question-by-question basis, employing both quantitative and qualitative approaches to address the research questions [59, 62]. Upon establishing the research questions, the investigation proceeded in three phases. In the initial phase, a questionnaire was developed, drawing upon both existing scholarly works and original questions aligned with the research questions identified. In the subsequent phase, this questionnaire was distributed to individuals actively engaged in the software architecture decision-making process, as well as to researchers possessing pertinent industry experience and insight. In the final phase, the gathered responses were methodically analyzed, both qualitatively and quantitatively, to draw conclusions. Each of these stages is elaborated upon further below.

3.2.2.1. First Phase: Design of Questionnaire

The survey's design was developed to delve into the complexities of the architectural decision-making process in software projects. Its main goal was to reveal the diverse factors that influence these decisions, the obstacles faced by practitioners, and the common methods for recording and rationalizing decisions. To accomplish this, we leveraged insights from our initial study [21] and combined them with extensive reviews of relevant literature. This method allowed us to devise a questionnaire that not only reflects the complex nature of architectural decision-making but also meets our research goals of pinpointing areas for improvement and understanding the effects of these decisions on software quality and team interactions.

To ensure the validity and reliability of the questionnaire, several methodological steps were undertaken. Initially, the survey questions were developed through a collaborative process

involving both academic researchers and industry experts, ensuring that they accurately reflect the real-world complexities of architectural decision making. Additionally, the reliability of the questionnaire was evaluated using Cronbach's Alpha, and the findings indicated satisfactory internal consistency across the various sections of the questionnaire. This thorough method in the design and evaluation of the questionnaire underpins the strength of our findings.

Our questionnaire is structured into four distinct sections, encompassing a total of 28 varied questions. Each section targets specific areas of interest. The initial segment gathers demographic data about the respondents. In the next section, the companies the participants were previously associated with and the projects they participated in are questioned. The third segment probes into the methods employed for making and documenting architectural decisions within the participants' most recent or current company/project. The final segment aims to capture the respondents' overall perspectives and understanding of software architecture, encouraging answers based on their comprehensive architectural knowledge and experiences. Further details on these sections and the specific questions they include are provided below, with the questions listed in Table 3.8, including the type of each question. Additionally, the table's rightmost column maps the survey questions to the corresponding research questions. The questionnaire, along with the anonymized responses, is available for online access [1].

Table 3.8 Survey questions. (OE: open-ended, MA: multiple answer, MC: multiple-choice, LS: Likert scale, Y/N: yes/no)

ID	Question	Type	Source	Condition	RQ
Questions about participants' profile					
Q1	Name- Surname	OE			
Q2	E-Mail	OE			
Continued on next page					

[1] https://doi.org/10.5281/zenodo.7515757

Table 3.8 – continued from previous page

ID	Question	Type	Source	Condition	RQ
Q3	What is the highest academic degree you have obtained? -Undergraduate -Graduate -Doctorate	MA			
Q4	What roles have you worked in software development projects throughout your career?	MC			
Questions about the companies and projects					
Q5	When you consider all the projects you have worked on until today, what is the maximum number of software engineers/software developers in the project with the most employees?	OE	[21]		
Q6	Which of the following software architectural patterns did you use in the past or current projects?	MA			
Q7	Which of the software architectural patterns you have chosen above do you have the most experience with?	MC			
Q8	Which types of software projects do you predominantly experience?	MA			

Continued on next page

Table 3.8 – continued from previous page

ID	Question	Type	Source	Condition	RQ
Q9	In the light of your experience in making architectural decisions in past projects, could you evaluate the influence of the following difficulties as 5 (very influential), 4 (influential), 3 (neutral), 2 (not so much influential), 1 (not influential at all), NE (Never Experiences)? F1-You received conflicting recommendations from various sources about which decision alternative to choose F2-There were no previous similar decisions to compare this decision against F3-It was hard to identify a superior decision alternative from the alternatives under consideration F4-The decision required a lot of thinking from you F5-It was hard to convince stakeholders to accept a certain decision alternative F6-Stakeholders had strongly diverging perspectives about the decision F7-You needed to influence some stakeholders without having formal authority over them F8-The decision had too many alternatives	LS	[4]		RQ2, RQ2.1

Continued on next page

Table 3.8 – continued from previous page

ID	Question	Type	Source	Condition	RQ
Q9	F9-The decision had too few alternatives F10-Analyzing alternatives for this decision took a lot of effort F11-Some quality attributes were considered too late in the decision-making process F12-Too many people were involved in making the decision F13-Dependencies with other decisions had to be taken into account F14-The decision had a major business impact F15-You had to respect existing architectural principles F16-Too little time was available to make the decision F17-You had a lot of peer pressure F18-There were many tradeoffs between quality attributes F19-You lacked experience as an architect F20-You lacked domain-specific knowledge (e.g. new customer) F21-More information was needed to reduce uncertainty when making the decision	LS	[4]		RQ2, RQ2.1
Questions about the companies they have worked with and the projects they have been involved in now and in the past (These questions collect information about company structures and the role of participants in their current or past company)					
Q10	Are you currently working in a company actively?	Y/N			

Continued on next page

Table 3.8 – continued from previous page

ID	Question	Type	Source	Condition	RQ
Q11	What is your role in the company you work for?	OE		Displayed only if Q10 is “Yes”	
Q12	How long have you been working in your company?	OE		Displayed only if Q10 is “Yes”	
Q13	How long have you been in your current role in your company?	OE		Displayed only if Q10 is “Yes”	
Q14	Approximately how many software engineers/software developers work in your company?	OE	[21]	Displayed only if Q10 is “Yes”	
Q15	What is the number of employees in the project you take part in (If you are involved in more than one project, answer based on the project largest in size)?	OE	[21]	Displayed only if Q10 is “Yes”	
Q16	How many people are involved in the role of ”software architect” in the project you take part in (If you are involved in more than one project, answer based on the project largest in size)?	OE		Displayed only if Q10 is “Yes”	
Q17	What is the software development lifecycle model used in your company (if more than one model is used, write them all)?	OE		Displayed only if Q10 is “Yes”	RQ2.1, RQ2.2
Questions about how architectural decisions are made in practice and how they are documented in participants’ current or last company/project (These questions collect information regarding decision-making activities and architectural decision documentation)					
	Are all or some of the architectural decisions in the company documented?	Y/N	[21]		RQ1, RQ2.1, RQ2.2
Q18	Continued on next page				

Table 3.8 – continued from previous page

ID	Question	Type	Source	Condition	RQ
	a. Which architectural decisions are documented?	OE	[21]	Displayed only if Q18 is “Yes”	RQ1
	b. Where are these documented architectural decisions stored?	OE	[21]	Displayed only if Q18 is “Yes”	RQ1, RQ1.2
	c. Which information is kept for each architectural decision? Please list down the major contents of the document.	OE		Displayed only if Q18 is “Yes”	RQ1, RQ1.2
	d. Are the documented and stored architectural decisions accessible to all developers? -Yes -No -Partially –Other	MC	[21]	Displayed only if Q18 is “Yes”	RQ1
	e. Do you have an architectural decision in your project or company that have any decisions been left undocumented intentionally?	Y/N	[21]		RQ1
	f. Do you follow the architectural decisions that affect each other in your project or company?	Y/N			RQ1 RQ1.3
Q19	Who usually makes the architectural decisions at your current or last company? -Decisions are made by only one person -Decisions are made by a team of more than one person	MC	[21]		RQ1, RQ2.1, RQ2.2

Continued on next page

Table 3.8 – continued from previous page

ID	Question	Type	Source	Condition	RQ
	a. How many people usually participate in team decision-making to create the software architecture?	OE		Displayed only if Q19 is "Decisions are made by a team of more than one person"	RQ1, RQ1.1
	b. What process(es) do you use most frequently for decision-making? - Brainstorming - Voting - Delphi - Consensus - Analytical Hierarchy Process (AHP)	MA	[53]	Displayed only if Q19 is "Decisions are made by a team of more than one person"	RQ1, RQ1.1
	c. Briefly describe the general structure and stakeholders of the team involved in your architectural decision-making process (e.g. architects, customers, business people, etc.).	OE	[53]	Displayed only if Q19 is "Decisions are made by a team of more than one person"	RQ1
General views and information about software architecture: These questions try to capture the general description of software architecture from practitioners' perspectives and to determine areas that need improvements					
Q20	How would you define the concept of "software architecture"?	OE	[21]		

Continued on next page

Table 3.8 – continued from previous page

ID	Question	Type	Source	Condition	RQ
Q21	What types of architectural decisions do you think are typically more critical to the project in practice? -Tool -Technology -Process (Development process-Methodology) -Organization (Business Environment) -Property Decisions (state the central qualities of a system and include design rules and guidelines, as well as constraints on a system) -Structural Decisions (lead to the creation of subsystems) -Behavioral Decisions (more related to how elements interact to address some functional or non-functional requirement.) -Other				

Continued on next page

Table 3.8 – continued from previous page

ID	Question	Type	Source	Condition	RQ
Q22	What factors do you think generally affect architectural decisions? Can you evaluate the following factors by scoring 5 (very effective), 4 (effective), 3 (neutral), 2 (less effective), 1 (ineffective)? -Management preferences -Tool and technology availability -Risk -Time -Quality attributes -User requirements -Maintenance / Product life cycle -Personal experiences/preferences -Business goals and strategies -Previous decisions (successful and unsuccessful) -Constraints -Costs	LS	[52]		RQ2
Q23	What are the factors that affect your architectural decision in terms of quality features? Among them, could you list the three most important quality factors?	OE	[21]		RQ2
Q24	How do you identify architecturally important requirements in the architectural decision-making process?	OE	[21]		RQ1
Q25	Even if you already have a solution in mind while making a decision, are you looking for alternative solutions to your needs?	Y/N	[21]		RQ3

Continued on next page

Table 3.8 – continued from previous page

ID	Question	Type	Source	Condition	RQ
Q26	How do you choose among alternative solutions?	OE	[21]		RQ3
Q27	How do you choose among alternative solutions?	OE	[21]		RQ3
Q28	Which of the following areas could be improved to make better architectural decisions? Do you have any additional areas to recommend? -Lightweight technique or tool to guide them -Improved documentation -Efficient information sharing -Keeping track of decisions and rationale -Making the decision-making more agile -The dependence of the decisions on each other and their traceability -Other	MA	[40]		RQ4

At the start of the questionnaire, we shared the goals of the survey, the intended audience, and the research questions with the participants. An overview of software architecture was also provided, as detailed below:

"The process of developing a Software Architecture requires extensive decision-making and the involvement of many stakeholders. To better control the software development process, the software project is guided by a software architecture selected from different architectural candidates. Software architecture is an architectural pattern that is constantly evolving as a result of a series of technical decisions. Based on the architectural candidates, the software architecture is shaped by the decisions taken to determine the most appropriate solution to meet the needs of the project. Accordingly, software architecture is an architectural model that constantly evolves as a result of a series of technical decisions."

After providing the introductory information, a short message was included to reassure participants about the confidentiality of their responses, guaranteeing that their personal information would remain private. Participants were then presented with a consent form, affirming their voluntary involvement in the study and their comprehension of its objectives.

3.2.2.2. Second Phase: Distribution of Questionnaire and Obtaining Responses

In the early stages of designing the survey, identifying respondents was our most important task. The distribution phase of our survey was strategically organized to target a broad and representative group of people interested in software architecture, both practitioners and researchers. We used multiple avenues to distribute the survey, including professional networks, social media channels and direct emails to industry experts. We motivated respondents to circulate the survey among their professional networks to increase the response rate and ensure a broad reach.

Our focus was on two key demographics: software practitioners actively engaged in or previously involved in the architectural decision-making process, and researchers possessing pertinent expertise and experience in the industry. Consequently, we distributed survey invitations to individuals involved in software architecture decision-making, including members of software architecture-related email lists and social media platforms (such as LinkedIn and Facebook), authors of scholarly articles on software architecture, and professionals labeled as ”software architects” on LinkedIn. For selecting paper authors, we specifically targeted those with contributions to journals or conferences focusing on software architecture. On LinkedIn, we reached out to individuals identified through searches for the term ”software architect”. The announcement requesting the participation of software architects in our study was posted in the following groups.

- Software Architecture [2]
- Software Architecture and Design [3]

[2]https://www.linkedin.com/groups/2967358/
[3]https://www.linkedin.com/groups/3740067/

- Software Architecture Architecting for Architects [4]
- Software Architects Discussions [5]

We dispatched over 1000 emails, LinkedIn, and Facebook messages directly to professionals and scholars in the field, and also reached out indirectly to many through announcements in groups, as well as posts on forums and social media platforms. Additionally, we encouraged practitioners to share the survey email with colleagues they knew who might be interested in participating. This approach led to a total of 101 individuals completing the survey. The survey was hosted online using JotForm and remained accessible from April 2021 through July 2021.

3.2.2.3. Third Phase: Analysis of Responses

In this part, we describe the methods used for collecting, organizing, and analyzing the survey responses. After concluding the survey, we applied both quantitative and qualitative techniques to process the responses. For quantitative analysis, we used statistical methods including frequency distribution, mean, and mode calculations. For qualitative responses, we conducted content analysis to identify thematic patterns and insights. Utilizing these combined approaches allowed us to derive thorough conclusions from the data, ensuring high validity and reliability in addressing our research questions. The survey was deployed online, with participant data compiled into a spreadsheet format. In this spreadsheet, questions were listed horizontally while participant responses were aligned vertically. We counted responses for binary (Yes/No), Multiple Choice, and Multiple Answer formats, excluding Likert Scale questions for which we calculated mode, median, and mean statistics.

The survey included two subsections aimed at evaluating the challenges in architecture and the factors influencing architectural decisions. The consistency of these subsections was verified using Cronbach's Alpha [63], with the architectural challenges subsection

[4]https://www.linkedin.com/groups/3767172/
[5]https://www.facebook.com/groups/127639125043271/

comprising 21 items (α = .724), the decision-influencing factors subsection containing 12 items (α = .764), and the combined subsections including 33 items (α = .773). With all three Cronbach's Alpha values falling between 0.7 and 0.8, the subsections and the survey as a whole were deemed reliable for internal consistency as per the criteria of George and Mallery [64].

Qualitative research methodologies seek to understand individuals' behaviors, beliefs, experiences, and attitudes [62], analyzing non-numeric data like text, video, and audio to extract and assess concepts or experiences. These methods aim to grasp the essence of research questions from a humanistic or idealistic standpoint, typically yielding qualitative data. Among these methods, content analysis systematically categorizes, quantifies, and interprets messages within qualitative data (e.g., text) both objectively and systematically, based on meaning and/or grammar [65]. Michael describes content analysis as a process of qualitative data reduction and interpretation aimed at identifying significant patterns and meanings [59]. In our study, we employed content analysis for the open-ended questions, segmenting the collected data into meaningful units to conceptually understand each segment. This process involved open coding to compare, contrast, and identify patterns within the data, thereby categorizing common themes [66] [67]. Subsequently, quantitative analysis was applied to these codes to identify recurring patterns and establish categories. These categories were then synthesized into broader themes, encapsulating generalized category insights. The codes and categories are available online [6]. Furthermore, we utilized the Mann-Whitney U test [68], a nonparametric test serving as an alternative to the two-sample t-test, to assess statistically significant differences between two distinct groups by analyzing their mean ranks.

[6]https://doi.org/10.5281/zenodo.7515757

3.3. ANALYSIS OF RESULTS

3.3.1. Participant Demographics

Table 3.9 displays the demographic information of the survey participants. All 101 respondents are currently employed in the industry or have previously been involved in making decisions related to software architecture in their professional lives. The participants in this study come from companies located in 27 different countries, with many based in Turkey, the United States, Canada, India, and others. A large portion of the respondents hold a master's degree (63%), followed by 19% with doctoral degrees, and 18% with undergraduate degrees. Over their careers, these individuals have held a variety of roles, including software architect, team leader, software developer, software tester, and project manager (Table 3.9).

Figure 3.1 illustrates the various roles held by survey participants over their careers and the breakdown of their current or most recent professional titles. Participants' roles were initially provided in an open-ended format, which we then categorized as shown in the figure. The data reveal that a significant majority have experience working as software architects, software developers, and team leaders. When looking at their most recent or current roles, software architects form the largest group, with 32 participants identifying with this position. We also inquired about the participants' current employment status, finding that 93 of them are actively employed in a company, while the remaining eight have had experience with software architecture decision-making in the past. Additionally, when asked about the presence of software architects in the projects they were involved in, only five participants reported the absence of a software architect role in their projects. In contrast, 28 indicated there was one software architect, and the majority (67) stated that their projects included two or more individuals in the software architect role.

Table 3.9 Demographics of participants

Unique respondents (N = 101)		
	Gender	
	n	%
Female	7	6.9
Male	94	93.1
	Country	
	n	%
Turkey	48	47.5
United States	11	10.9
Canada	3	3.0
India	3	3.0
Other	36	35.6
	Education	
	n	%
BSc	18	17.8
MSc	63	63.4
PhD	19	18.8
	Work Experience on Current Position	
	n	%
Less than a year	13	14.0
1 - 5 years	62	66.7
6 - 10 years	13	14.0
11 - 20 years	4	4.3
More than 20 years	1	1.1

3.3.2. Company and Project Demographics

In this part of the survey, we asked participants regarding their previous project involvement, including the type of these projects, the software patterns they are most familiar with, and the challenges encountered during the architectural decision-making stages of these projects. Based on responses to questions about the types of projects they most commonly worked on, it was discovered that a significant portion of the participants had extensive experience in

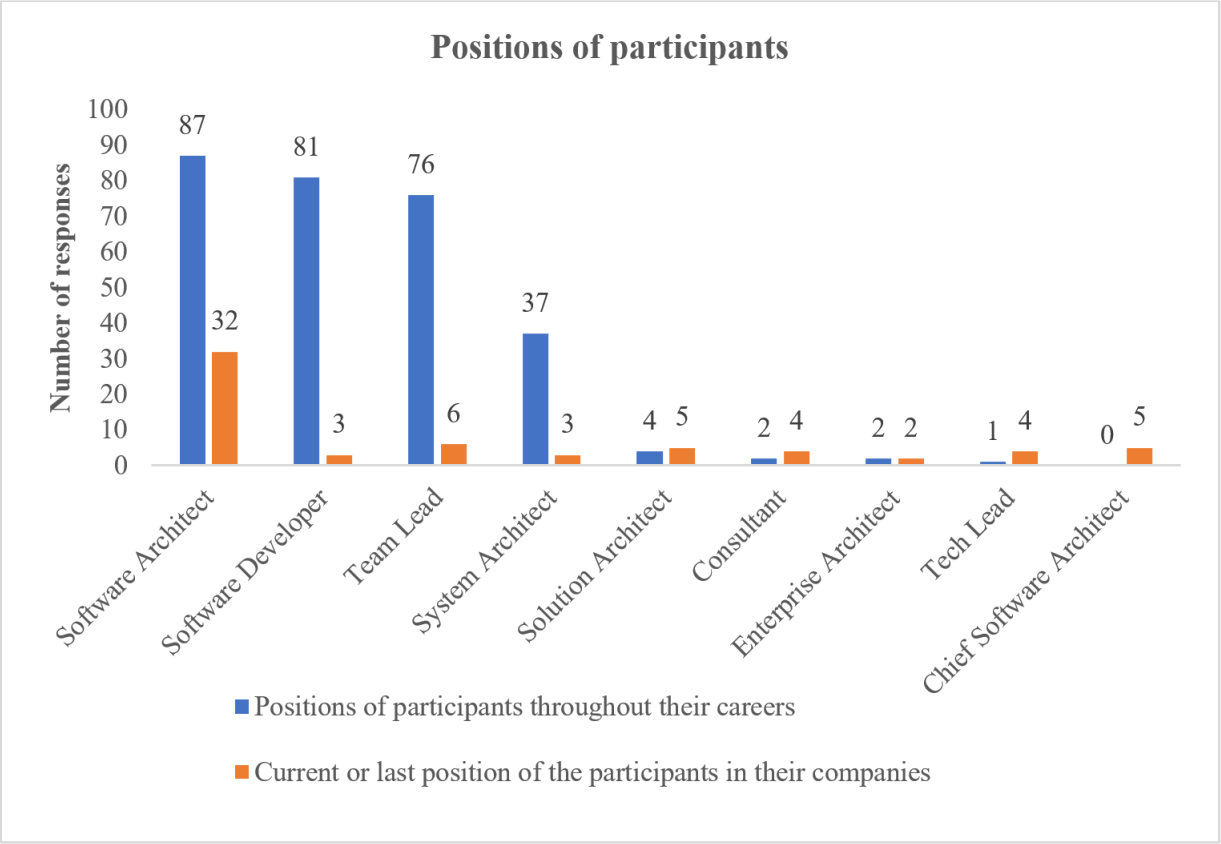


Figure 3.1 Positions of participants throughout their careers and current or last positions of the participants in their companies

developing business application software (32%) and web applications (31%), as depicted in Figure 3.2.

Participants were asked about the architectural patterns they employed in previous projects. A range of architectural patterns was provided for selection, and participants had the option to specify any design patterns not listed. The occurrence of each pattern chosen was tallied and is displayed in Figure 3.3, revealing that service-based architecture (chosen by 92 participants) and layered architecture (chosen by 91 participants) emerged as the top preferences. Additionally, respondents indicated that they possess the most experience with layered architecture among the architectural patterns.

Participants were asked to assess and rank the scenarios listed in the first column of Table 3.10, based on the challenges they have previously encountered. They were to choose “I

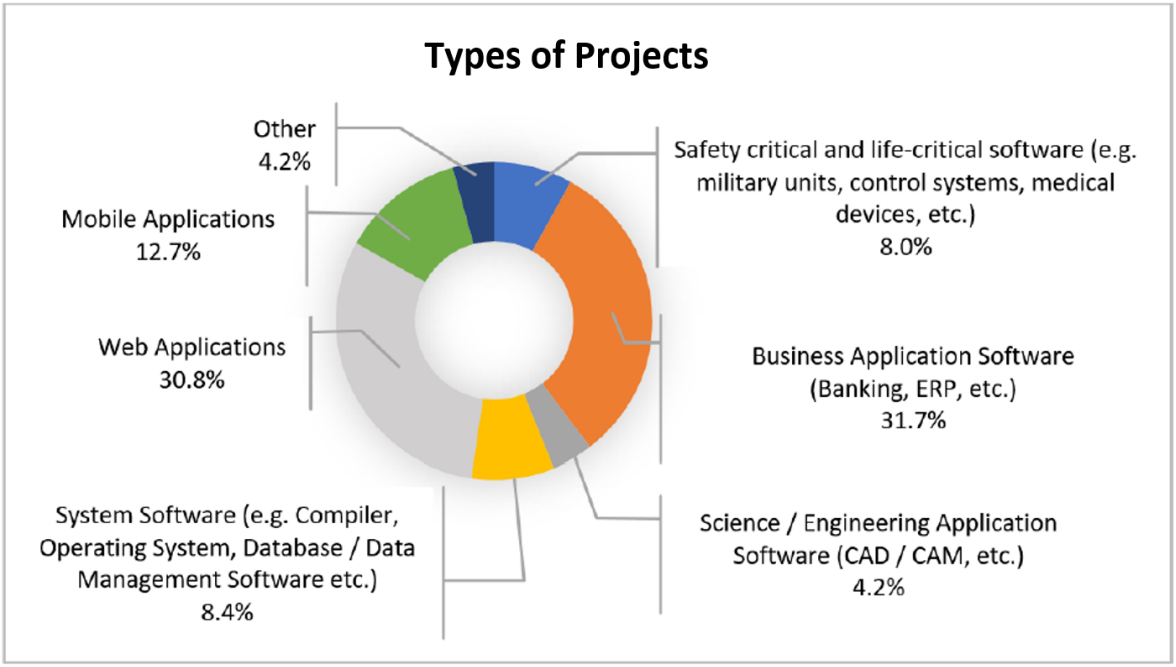


Figure 3.2 Types of projects that the participants have most experienced

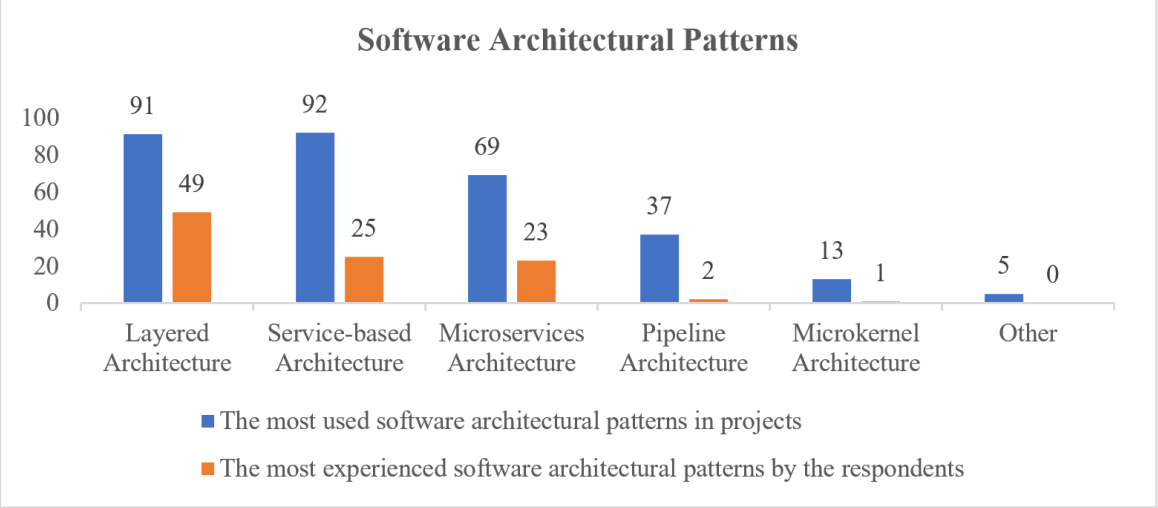


Figure 3.3 Types of architectural patterns that participants have most experienced

completely agree" (5) if a particular situation presented significant difficulty, and "I do not agree at all" (1) if the situation had not been challenging in the past. For scenarios they had never encountered, they selected the "never experienced" option. The mode, median, and mean values for the responses to these questions are provided in the table, excluding the "never experienced" responses from these calculations.

To address research question RQ2.1, we conducted a Mann-Whitney U test [69], a non-parametric test that serves as an alternative to the two-sample t-test. The responses to question Q9 were categorized based on:

- whether architectural decisions are documented (N: 79) or not (N: 22),
- whether the development life-cycle model used is Agile (N: 72) or not (N: 29),
- whether decisions are made individually with a single decision-maker (N: 9) or as a group (N: 91),
- whether architectural decisions are verified (N:28) or not (N:40),
- whether the participants are experienced with Business Application Software (N:75) or not (N:26),
- whether the participants are experienced with Web Applications (N:72) or not (N:29),
- whether the participants are experienced with Layered Architecture (N:49) or not (N:51),
- whether the participants are experienced with Service-based Architecture (N:25) or not (N:75),
- whether the participants are experienced with Microservices Architecture (N:23) or not (N:77).

We raised the following hypotheses with respect to the groups above:

- H1_1: Technical and social challenges on decision-making (Q9) are related to whether architectural decisions are documented (N: 79) or not (N: 22),
- H1_2: Technical and social challenges on decision-making (Q9) are related to the development lifecycle model used is Agile (N: 72) or not (N: 29),

- H1_3: Technical and social challenges on decision-making (Q9) are related to whether decisions are made individually with a single decision-maker (N: 9) or as a group (N: 91),
- H1_4: Technical and social challenges on decision-making (Q9) are related to whether architectural decisions are verified (N:28) or not (N:40),
- H1_5: Technical and social challenges on decision-making (Q9) are related to whether the participants are experienced with Business Application Software (N:75) or not (N:26),
- H1_6: Technical and social challenges on decision-making (Q9) are related to whether the participants are experienced with Web Applications (N:72) or not (N:29),
- H1_7: Technical and social challenges on decision-making (Q9) are related to whether the participants are experienced with Layered Architecture (N:49) or not (N:51),
- H1_8: Technical and social challenges on decision-making (Q9) are related to whether the participants are experienced with Service-based Architecture (N:25) or not (N:75),
- H1_9: Technical and social challenges on decision-making (Q9) are related to whether the participants are experienced with Microservices Architecture (N:23) or not (N:77).

From the analysis, the most daunting challenge in the architectural decision-making process was identified as F14 – "The decision had a significant impact on the business," which scored a mode of 5, a median of 4, and an average score of 4.19. On the other hand, F9 – "The decision had too few alternatives" emerged as the least problematic. These findings are in line with those from the research conducted by Tofan et al. [4]. For each identified challenge, the number of participants who indicated they had "never experienced" the situation is listed in the "NE" column of Table 3.10. A high-resolution version of Table 3.10 is made available for detailed examination [7]. p values obtained with the Mann-Whitney U test. p values less than the 0.1 significance level and Values less than 0.05 are shown in bold.

[7] https://doi.org/10.5281/zenodo.7519857

Table 3.10 Relationships of technical and social challenges in decision making for hypothesis group 1 (H1)

Factors	Scores				p values (Mann-Whitney U)							
	Average	Mode	Median	NE	H1_1	H1_2	H1_3	H1_4	H1_5	H1_6	H1_7	H1_8
F14 – The decision had a major business impact	4.19	5	4	0	0.610	0.477	0.902	0.892	**0.087**	0.363	0.217	0.671
F13 – Dependencies with other decisions	4.06	4	4	0	0.151	0.881	**0.090**	0.209	0.411	0.793	0.646	0.861
F21 – More information was needed	3.80	4	4	1	0.664	0.324	0.101	0.265	**0.081**	0.587	0.104	0.189
F15 – Respect for existing architectural principles	3.75	4	4	1	0.930	0.641	0.307	0.588	0.114	0.650	0.506	0.821
F4 – The decision required a lot of thinking from you	3.71	4	4	1	0.927	0.225	0.959	0.784	0.447	**0.072**	0.927	0.597
F7 – You needed to influence some stakeholders without having formal authority over them	3.67	4	4	0	**0.043**	0.266	0.135	0.859	0.651	0.246	0.971	0.790
F6 – Stakeholders had strongly diverged perspectives about the decision	3.58	4	4	2	0.111	0.716	0.319	0.818	0.484	0.228	0.865	0.194
F16 – Too little time was available to make the decision	3.52	4	4	0	0.800	0.763	0.406	0.423	0.795	0.156	**0.007**	**0.004**
F18 – There were many tradeoffs between quality attributes	3.52	4	4	1	0.872	0.919	0.371	0.512	**0.049**	0.804	0.438	0.078
F5 – It was hard to convince stakeholders to accept a certain decision alternative	3.47	4	4	1	0.104	0.931	**0.005**	**0.085**	0.971	0.815	0.953	0.410
F1 – You received conflicting recommendations from various sources about which decision alternative to choose	3.45	4	4	2	**0.091**	0.804	**0.007**	0.123	0.719	0.363	0.841	0.356
F11 – Some quality attributes were considered too late in the decision-making process	3.40	4	4	0	0.151	0.580	0.127	0.990	0.408	0.671	**0.037**	**0.009**
F10 – Analyzing alternatives for this decision took a lot of effort	3.35	4	4	0	**0.055**	0.697	0.860	**0.048**	0.935	0.656	0.439	0.973
F2 – There were no previous similar decisions to compare this decision against	3.33	4	3	0	0.769	0.928	0.415	0.498	0.810	0.555	0.340	0.783
F12 – Too many people were involved in making the decision	3.26	4	3	0	**0.009**	0.320	**0.065**	0.939	0.262	0.304	0.766	0.582
F3 – It was hard to identify a superior decision alternative from the alternatives under consideration	3.22	4	3	1	0.145	0.778	0.356	0.143	0.730	0.565	0.258	0.10
F20 – You lacked domain-specific knowledge (e.g., new customer)	3.02	4	3	4	0.679	0.334	0.856	0.365	0.146	0.774	0.515	0.169
F8 – The decision had too many alternatives	3.00	4	3	2	0.530	0.249	0.682	0.558	0.568	0.546	0.432	0.786
F17 – You had a lot of peer pressure	2.92	2	3	3	0.430	**0.008**	**0.078**	0.238	0.203	**0.034**	**0.001**	**0.034**
F19 – You lacked experience as an architect	2.66	2	3	2	0.791	0.836	0.929	0.674	0.108	0.236	0.264	0.449
F9 – The decision had too few alternatives	2.54	2	2	3	0.231	0.987	0.597	0.761	0.335	0.827	0.351	0.338

The p-values were calculated for each group, with the significance threshold set at 0.10. Although the p-value criterion for significance was 0.10, the table highlights p-values smaller than 0.05 in a deeper shade of gray. The data indicate that for the hypothesis H1_9, concerning the challenges faced during the architectural decision-making process, no statistically significant findings were observed, and thus these results were not included in Table 3.10. When examining the various hypotheses connected with different groups against the backdrop of social and technical challenges, a substantial number of hypotheses, particularly those related to F17 (including H1_1, H1_2, H1_3, H1_6, H1_7, and H1_8), were corroborated. This means there's a statistically noticeable disparity between the mentioned group pairs and how they perceive these challenges, with different perceptions based on the challenge level. Table 3.10 displays that for factors like F1 and F12, a statistically significant variation exists for both the documentation of architecture decisions and the size of the decision-making group. Responses for F13, F5, F1, F12, and F17 showed a significant difference between decisions made by an individual and those made by a collective. These difficult scenarios are commonly associated with stakeholder involvement, and the distinct perceptions between group and individual decision-makers underscore this point.

Concerning the software lifecycle methods applied within their organizations, participants were prompted with an open-ended question. They were advised to list all methods if multiple were in use. Three individuals did not respond to this inquiry, and one provided an irrelevant response. The contributions from the remaining 97 participants are visualized in Figure 3.4. Some participants referenced a broad category like "agile," while others named specific models such as "scrum." In the analysis, each distinct method was treated as a separate category. Methods delineated with commas or dashes, like "Agile, Waterfall" or "Agile-Waterfall," were regarded as separate methodologies. The Agile approach was mentioned by a total of 90 respondents. While 44 mentioned "agile" in a general sense, the rest specified particular agile frameworks they employ, including Scrum (35), Kanban (7), and Safe Agile (4).

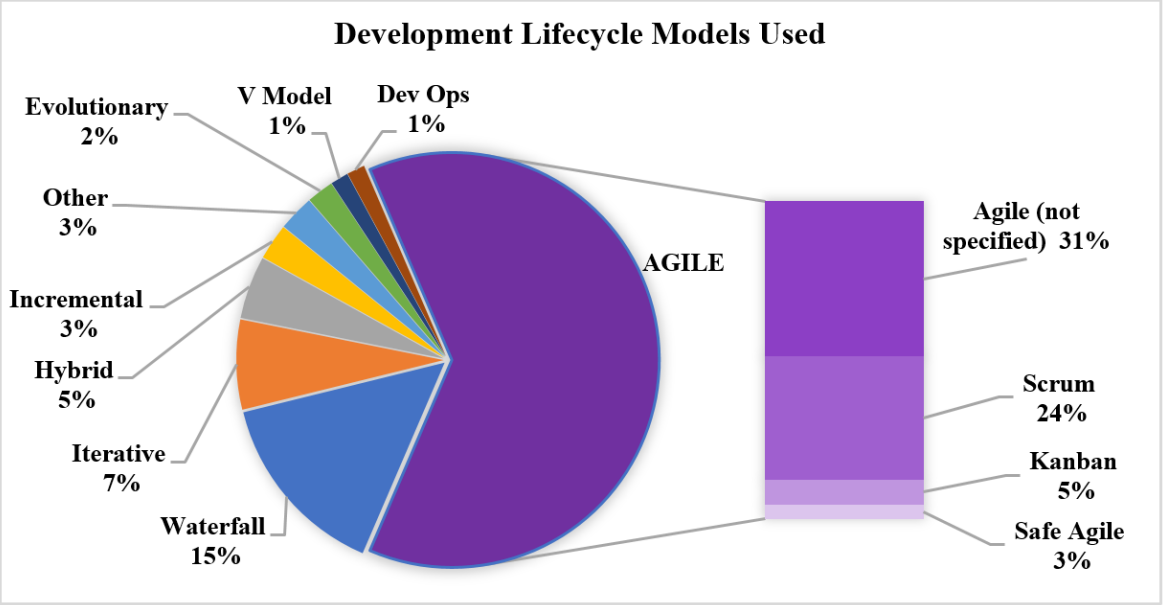


Figure 3.4 Development Lifecycle Models used by companies of the participants

3.3.3. Results from questions about how architectural decisions are made in practice and how these decisions are documented in participants' current or last company/project

In this segment, inquiries focused on the procedural aspects of making architectural decisions, the documentation of these decisions, the specifics contained within these documents, their storage locations, and the composition of the teams making these decisions. Regarding the question, "Are the company's architectural decisions documented in any form?", 79 respondents affirmed while 22 negated. Those confirming were further questioned about the nature of decisions documented, the storage locations of these documents, their contents, and their accessibility to all developers. When asked if all developers could access the architectural decision documentation, 61 indicated "yes", 16 said "partially", one responded "no", and one did not provide an answer. The respondents who noted that decisions were documented were also asked about the types of decisions documented. The documented architectural decision types are listed in Table 3.11. A majority mentioned documenting all decisions made. Based on other responses, decisions frequently documented encompassed "services, tools, and technologies" such as the choice

of software languages for the project, application server requirements, database selections, and ”system-level decisions” covering the system’s architecture, the identification of architectural components, and their interrelations. Among these responses, one participant’s answer was incomplete and incorrect, and 23 respondents opted not to answer, with these responses being omitted from the analysis and not categorized.

Table 3.11 Responses to questions about architectural decisions that are documented

Q18. A. Which architectural decisions are documented?		
Responses	Frequency	Percentage
All Decisions	17	16.3%
Services, tools, and technologies	14	13.5%
System level decisions	12	11.5%
Architectural components	9	8.7%
High-level design decisions	8	7.7%
Significant evolution or change of architecture	6	5.8%
Key / Main decisions	5	4.8%
Principles	4	3.8%
Diagrams	4	3.8%
Infrastructural decisions	4	3.8%
Processes	4	3.8%
Selected design patterns	4	3.8%
Solution level decisions	3	2.9%
Strategies	2	1.9%
Policy on what to document is unclear	2	1.9%
Requirements	2	1.9%
Other	4	3.8%

We asked about the specific content maintained within decision documents. Table 3.12 lists the responses and their occurrence. The most frequently mentioned elements in decision documentation include options, trade-offs, and visual diagrams. Figure 3.5 illustrates the most prevalent types of content documented. The analysis indicates that there’s a notable frequency in documenting both alternatives and trade-offs.

Typically, decision documentation is housed in collaborative tools like Confluence, Wiki, GitHub, and JIRA, which facilitate team cooperation and information sharing. In the realm of software development, file-based documentation is a common method for capturing architectural knowledge [70], as reflected in the participant responses presented in Figure 3.6.

We queried participants on whether they track interrelated architectural decisions within their projects or organizations. Of the respondents, 78 indicated they do monitor such decisions, while 24 said they do not. Those who do track interdependent decisions reported they usually do so through regular documentation reviews and meetings.

Table 3.12 Major contents of architectural documents

Q18. C. Which information is kept for each architectural decision? Please list down the major contents of the document.		
Responses	Frequency	Percentage
Alternatives	19	10.2%
Tradeoffs	18	9.7%
Diagrams	16	8.6%
Context (Discussions) and problem statements	15	8.1%
Rationale of decisions	14	7.5%
Effects	10	5.4%
Decision	9	4.8%
Consequences	9	4.8%
Requirements	9	4.8%
Meeting notes	6	3.2%
Related decisions	5	2.7%
Design principles	5	2.7%
Date	4	2.2%
Decision drivers	4	2.2%
Costs	4	2.2%
Decision maker	4	2.2%
Status	3	1.6%
Domain	3	1.6%
Current Situation	3	1.6%
Title	2	1.1%
Constraints	2	1.1%
Risks	2	1.1%
Other	20	10.8%

Regarding the query "*What process(es) do you use most frequently for decision-making?*" 89 participants reported that architectural decisions are typically made collaboratively by a team comprising multiple individuals. On average, these decision-making teams consist of five members. Conversely, the remainder of the survey participants noted that decisions are made by a sole individual. We introduced several group decision-making procedures to those

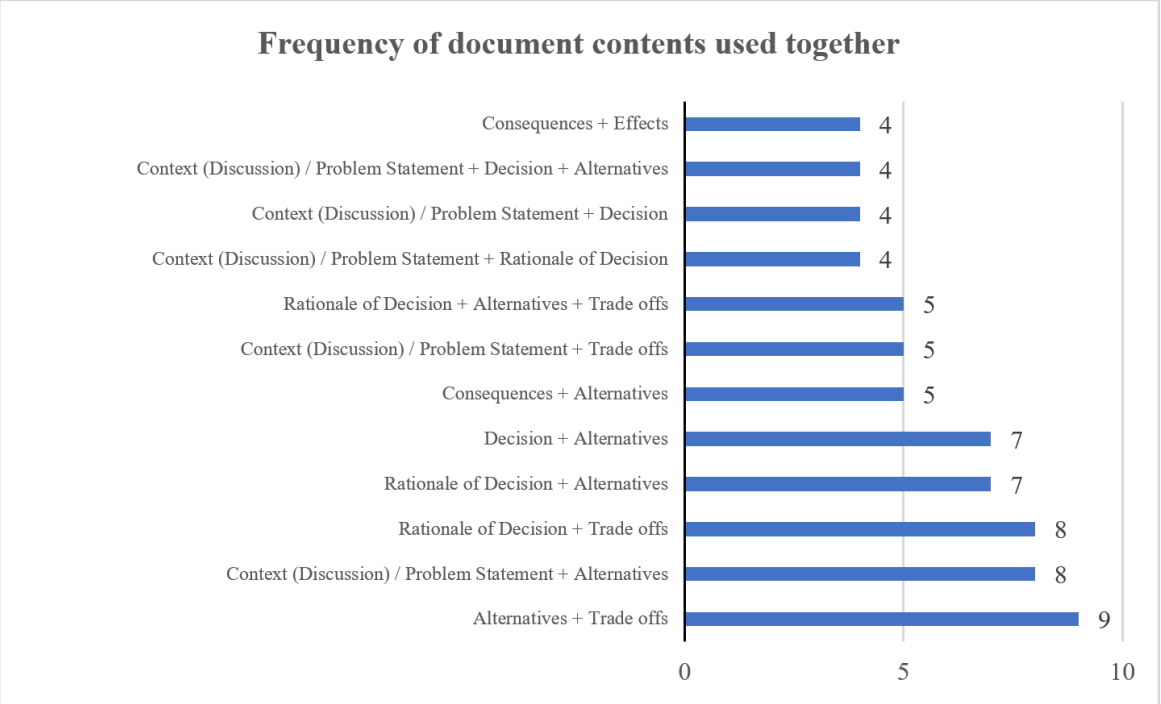


Figure 3.5 Frequency of document contents used together

who identified with the multi-person team approach, prompting them to select their most frequently employed method(s) through multiple-choice questions. In evaluating the survey data, we tallied the number of times each method was mentioned (such as Brainstorming, Consensus, AHP, Voting, Delphi, No formal method, and Other), and these tallies are depicted in Figure 3.7 (a). The results demonstrate a preference for brainstorming and consensus as the predominant decision-making techniques. Participants described a process of discussing and weighing different options to achieve a mutual agreement, a finding that aligns with observations made by Groher et al. [71]. Moreover, as depicted in Figure 3.7 (b), 22% of respondents adhere to a singular decision-making process (%23 Brainstorming, %15 Consensus, %2 No formal method). Among those who apply multiple strategies, a combination of brainstorming and consensus was the most frequently selected. Teams engaging in collective decision-making, as per the responses, can vary in size up to 20 members. Analysis of responses to an open-ended question about the general composition of the team and its stakeholders involved in decision-making suggests that such groups are predominantly made up of software architects and developers.

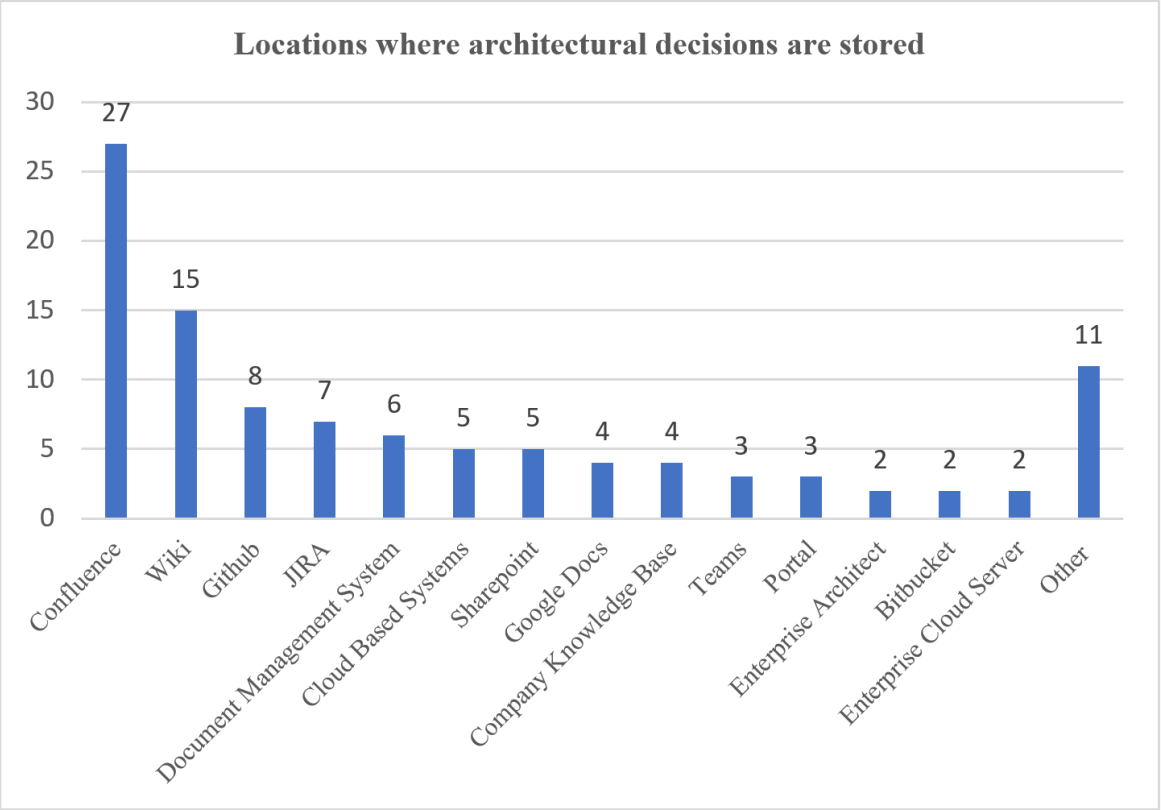


Figure 3.6 Places where architectural decisions are stored

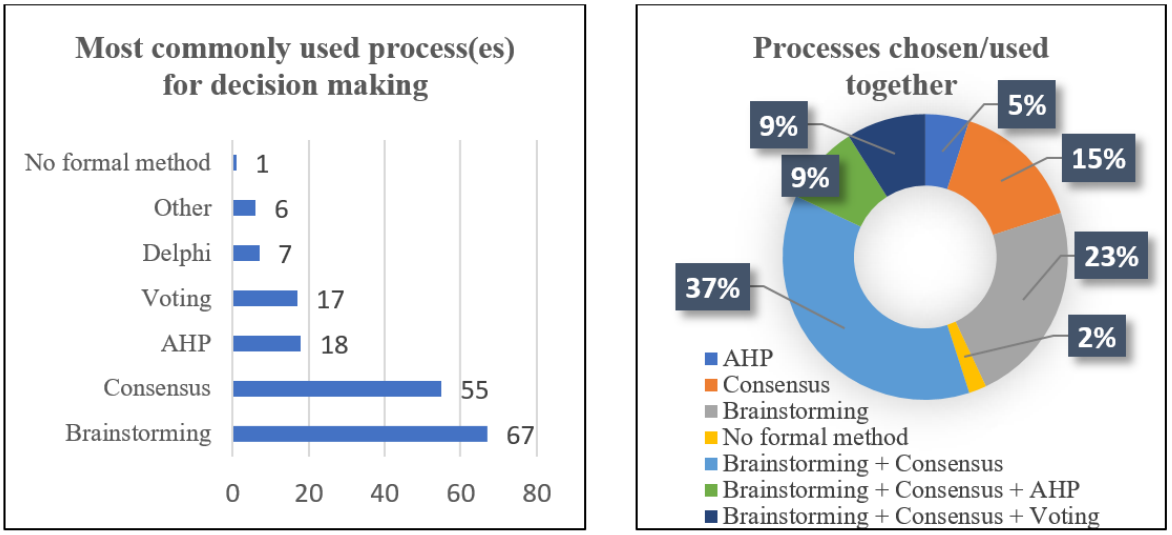


Figure 3.7 (a) Most commonly used process(es) for decision-making, (b) Number of process(es) chosen/used together

3.3.4. Results from general questions about software architecture

There is no unanimously accepted definition of software architecture either in the literature or among architects. Often, the terms software architecture and software design are observed to be used somewhat synonymously. Consequently, we asked participants to articulate their definitions of software architecture through written descriptions. Varied definitions were provided by the respondents. Exemplary responses are shared below. Common to most participant definitions is the conceptual framework for software architecture as delineated by Bass et al. [3], which is articulated in the introductory segment. Participants consistently characterized software architecture as the components of a program or system that satisfy requirements, along with the configuration of these components and the relationships among them. In Figure 3.8, we highlight the key terms and phrases extracted from the participants' definitions of software architecture. Additionally, here are some representative answers provided by the participants:

- P10: *"Organization of responsibility in order to convey the system's purpose and capability."*
- P15: *"The set of rules and recommended patterns that govern the growth of a piece of software."*
- P34: *"A living cell organism influencing every part of the system to work efficiently."*
- P95: *"Everything in Software Architecture is a tradeoff."*

We offered the participants a selection of architectural decision categories and requested them to identify the ones they consider most crucial in practice. The distribution of choices among these decision categories is depicted in Figure 3.9. Based on the feedback, a significant number of participants view decisions concerning architecture structure, technology-related decisions, process-oriented decisions, decisions about properties, and decisions influencing behavior as particularly vital for project success in a practical setting.

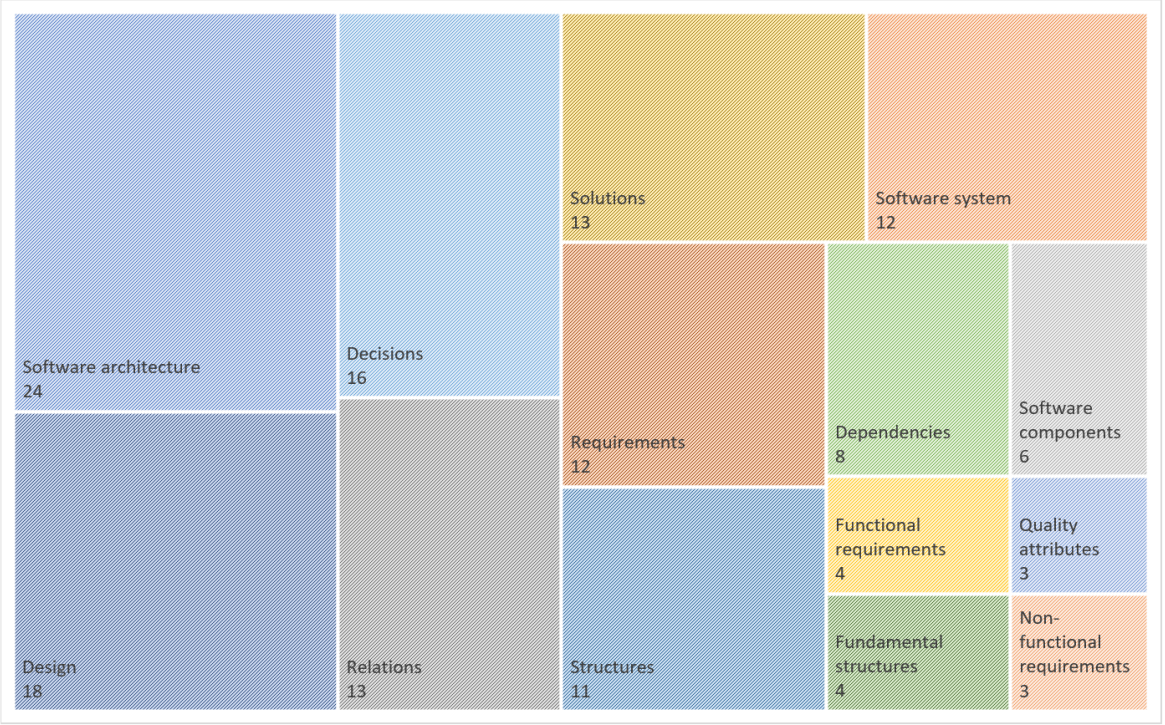


Figure 3.8 Keywords from participants' definition of software architecture

There can be many factors that affect architectural decisions in the architectural decision-making process [52]. We asked participants to evaluate a subset of these elements, including management preferences, the availability of tools and technologies, risks, time constraints, and quality attributes, rating their impact on a scale from 1 (ineffective) to 5 (very effective). As detailed in Table 3.13, "costs" emerged as the most impactful element, garnering an average score of 4.26. Beyond these specified factors, participants also highlighted the skills of the team, organizational policies, and the socio-cultural context as influential determinants.

To address research question RQ2.2, we conducted a Mann-Whitney U test [69] analyzing the data collected from the responses to question Q22. The data was categorized as follows:

- whether architectural decisions are documented (N: 79) or not (N: 22),
- the development lifecycle model used is Agile (N: 72) or not (N: 29),

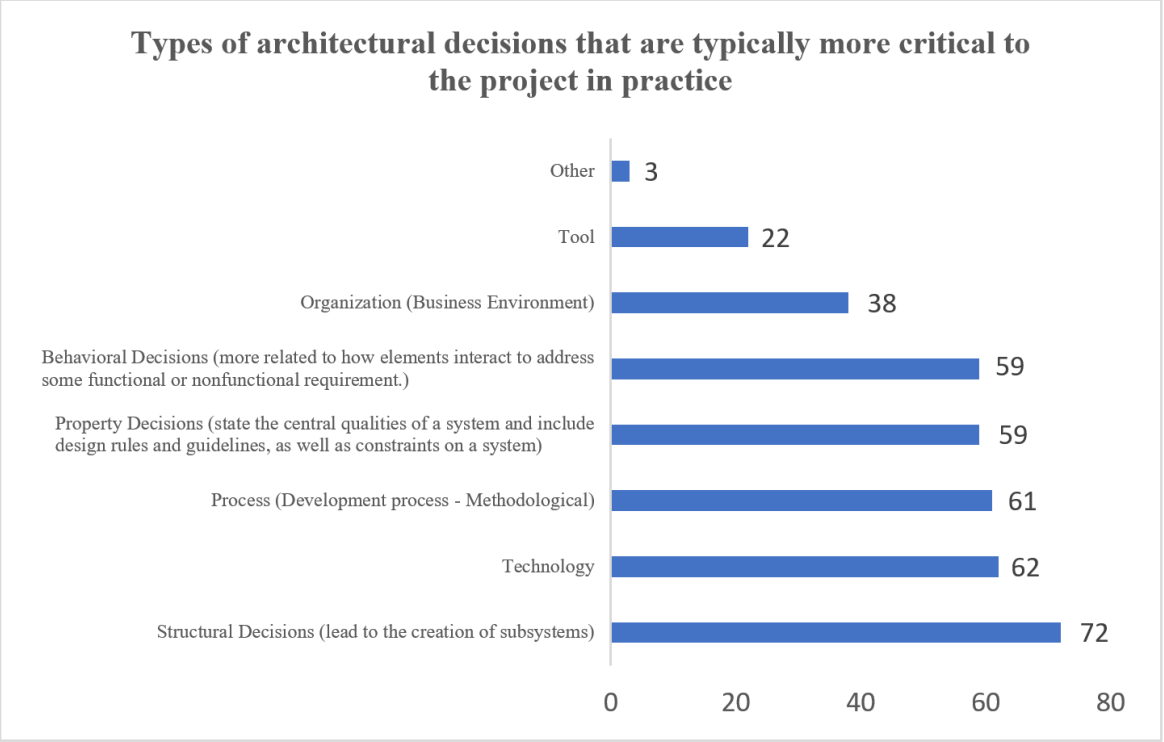


Figure 3.9 Types of architectural decisions that the participants think are more critical to the project in practice

- whether decisions are made individually with a single decision-maker (N: 9) or as a group (N: 91),
- whether architectural decisions are verified (N:28) or not (N:40),
- whether the participants are experienced with Business Application Software (N:75) or not (N:26),
- whether the participants are experienced with Web Applications (N:72) or not (N:29),
- whether the participants are experienced with Layered Architecture (N:49) or not (N:51),
- whether the participants are experienced with Service-based Architecture (N:25) or not (N:75),
- whether the participants are experienced with Microservices Architecture (N:23) or not (N:77).

We raised the following hypotheses with respect to the groups above:

- H2_1: Factors that affect architectural decisions (Q22) are related to whether architectural decisions are documented (N: 79) or not (N: 22),
- H2_2: Factors that affect architectural decisions (Q22) are related to whether the development lifecycle model used is Agile (N: 72) or not (N: 29),
- H2_3: Factors that affect architectural decisions (Q22) are related to whether decisions are made individually with a single decision-maker (N: 9) or as a group (N: 91),
- H2_4: Factors that affect architectural decisions (Q22) are related to whether architectural decisions are verified (N:28) or not (N:40),
- H2_5: Factors that affect architectural decisions (Q22) are related to whether the participants are experienced with Business Application Software (N:75) or not (N:26),
- H2_6: Factors that affect architectural decisions (Q22) are related to whether the participants are experienced with Web Applications (N:72) or not (N:29),
- H2_7: Factors that affect architectural decisions (Q22) are related to whether the participants are experienced with Layered Architecture (N:49) or not (N:51),
- H2_8: Factors that affect architectural decisions (Q22) are related to whether the participants are experienced with Service-based Architecture (N:25) or not (N:75),
- H2_9: Factors that affect architectural decisions (Q22) are related to whether the participants are experienced with Microservices Architecture (N:23) or not (N:77).

For each specified group, we derived p-values based on responses concerning factors influencing architectural decisions, the presence of documentation for these decisions, the developmental lifecycle models applied, and the decision-making team sizes. Shifts in the perception of architectural decision influencers (Question 22) were noted according to the documentation status of these decisions, the development models employed, and the mode

Table 3.13 Relationships of factors that the participants think they affect architectural decisions for hypothesis group 2 (H2)

What factors do you think generally affect architectural decisions? Can you evaluate the following factors by scoring 5 (very effective), 4 (effective), 3 (neutral), 2 (less effective), 1 (ineffective)?									
Factors	Avg.	Mode	Median	H2_2	H2_4	H2_5	H2_6	H2_7	H2_9
Costs	4.26	4	4	0.317	0.824	0.636	0.143	0.197	0.409
Quality attributes	4.25	4	4	0.662	0.146	0.329	0.280	0.857	0.703
User requirements	4.18	5	4	0.740	0.465	0.795	0.536	0.538	0.545
Constraints	4.17	4	4	0.237	0.852	0.643	0.780	0.588	0.944
Time	4.16	5	4	**0.097**	0.784	0.691	**0.065**	0.336	0.384
Business goals and strategies	4.16	5	4	0.131	**0.008**	**0.026**	0.994	0.278	0.561
Risk	4.11	4	4	0.527	0.176	**0.037**	0.624	0.585	0.893
Previous decisions (successful and unsuccessful)	4.08	4	4	0.819	0.347	0.947	0.737	0.614	0.353
Tool and technology availability	3.87	4	4	0.934	0.377	0.563	0.844	0.797	0.233
Maintenance / Product lifecycle	3.85	4	4	0.587	0.931	0.784	0.440	0.245	0.257
Personal experiences/preferences	3.52	4	4	0.625	0.646	0.395	0.130	0.826	0.819
Management preferences	3.32	4	3	0.109	0.574	0.111	0.850	**0.032**	**0.016**

Note: p values obtained with the Mann-Whitney U test. p values less than the 0.10 significance level are shaded. Values less than 0.05 are also shown in bold.

of decision-making (individual vs. group). In this analysis, the threshold for statistical significance was set at 0.10, with evaluations made accordingly; however, instances with p-values under 0.05 were distinctly highlighted in darker gray within Table 6. The analysis revealed no significant outcomes for hypotheses H2_1, H2_3, and H2_8, which relate to the impact on architectural decisions, thus these findings were omitted from Table 3.13. Furthermore, p-values were computed across groups differentiated by the verification of architectural decisions (verified: N=29; not verified: N=36), software project types (Business Application Software: N=75; other software projects: N=26; Web Applications: N=72; other types: N=29), and architectural patterns (Layered Architecture: N=91; others: N=10; Service-based Architecture: N=92; others: N=9; Microservices Architecture: N=68; others: N=33), with these results presented in Table 3.13. From this table, it was discerned that hypothesis H2_5, which concerns business goals, strategies, and risk, was substantiated, indicating a statistical discrepancy in decision-making factors between users of business application software versus other software types. The study found no hypothesis support for the influence of tools and technology availability, time constraints, quality attributes, user needs, maintenance or product lifecycle considerations, personal preferences or experiences, previous decisions, limitations, or costs on architectural decision-making.

Additionally, we explored the presence of any statistically meaningful connections between the groups detailed in Tables 3.10 and 3.13, as well as their internal correlations. The Spearman's rank-order correlation method was applied to assess these relationships. The outcomes are displayed in a matrix format within Table 3.14, showcasing repeated correlations and focusing solely on results derived from significant p-values. The assessment criterion was set at a p-value threshold of 0.05, indicating a statistically significant correlation for p-values below this threshold, and a lack of significant correlation otherwise. These findings are represented through p-values, with cells highlighted in green denoting significance for easier interpretation. According to Table 3.14, notable correlations are observed between Q9 and Q22, as well as within Q9 and Q22 themselves. Analysis of inter-correlations among responses to question 9, a Likert scale query, reveals that items F1, F17, and F21, each associated with ten distinct challenges, exhibit the most significant correlations. In the case of Q22, risk, time, quality attributes, user requirements, maintenance, business goals, and costs are identified as having a higher incidence of significant correlations compared to other factors. Specifically, the analysis between Q9 and Q22 indicates that F5 and F17 show statistically significant correlations with numerous factors outlined in Q22. Furthermore, considering the factor "Time" within Q22, it significantly correlates with a range of challenging situations outlined in Q9, encompassing eight distinct difficulties. An in-depth review of both Q9 alone and its comparative analysis with Q22 reveals that F17 demonstrates a statistically significant variance with the highest number of significant correlations (15) as noted in the "F17" row. Similarly, the "Time" row, with twelve significant correlations, indicates that the "Time" factor significantly differs from many other factors in terms of statistical significance.

In any given system, decision-makers are tasked with considering and fulfilling quality attributes through deliberate choices, such as selecting the optimal combination of options [55, 72]. The significance of software quality in the realm of software architecture decision-making has been underscored by Haoues and colleagues [73]. Quality attributes are pivotal in driving the design process towards the realization of high-quality systems, thereby necessitating their incorporation into decision-making and documentation processes

[74]. The critical nature of quality attributes was further highlighted by responses to question Q22, where participants were queried about the impact of quality metrics on architectural decisions, prompting them to identify up to three key metrics. Responses varied, with some participants mentioning only one metric, while others cited more than three. Analysis of these responses led to the categorization based on the ISO/IEC 25010 product quality model [75], with non-standard metrics classified as "other" and non-applicable responses as "unrelated responses", following expert review. Responses also included specific sub-attributes, which align with the eight quality characteristics defined in ISO/IEC 25010. Figure 3.10 reveals that maintainability and related sub-attributes were most frequently mentioned by 43 respondents, followed by performance and reliability, cited by 34 and 23 participants respectively. Metrics outside the ISO/IEC 25010 scope, like extensibility and scalability, were categorized as "other". Additionally, 27 participants provided responses deemed irrelevant to the query. While the findings from the quality metrics question may not be universally applicable across domains, the study by Bi et al. [76], which explored the connection between architectural patterns and quality metrics through an analysis of Stack Overflow discussions, suggests a correlation between quality metrics and architectural patterns. This study, similar to Bi et al.'s findings, indicates a discernible link between architectural patterns and quality metrics, with maintainability, performance, and security emerging as the most frequently mentioned metrics. Layered and service-based architectures were identified as the predominant architectural patterns, mirroring Bi et al.'s observation that performance and security, alongside performance and maintainability, are crucial quality metrics in discussions surrounding these architectural styles.

Based on the responses to the inquiry regarding the identification of key architectural requirements, it was not possible to define a distinct category. However, from the feedback collected, it was evident that experience plays a significant role in pinpointing these essential requirements. Alongside experience, engaging with stakeholders and conducting risk assessments were also cited as critical methods for recognizing architecturally significant requirements. Selected responses are summarized as follows:

- P4: *"We have no formal method other than experience."*
- P29: *The review of new system requirements and respective solutions that have: Impacts on the integrity of the architecture and opportunities to enhance the architecture"*
- P50: *"By starting at "what the software is supposed to do" and then drilling down."*
- P67: *"We do not use a methodology. By reviewing known patterns, architectural decisions of similar applications or technical reports, we base common solutions to meet general requirements and customize them if necessary."*

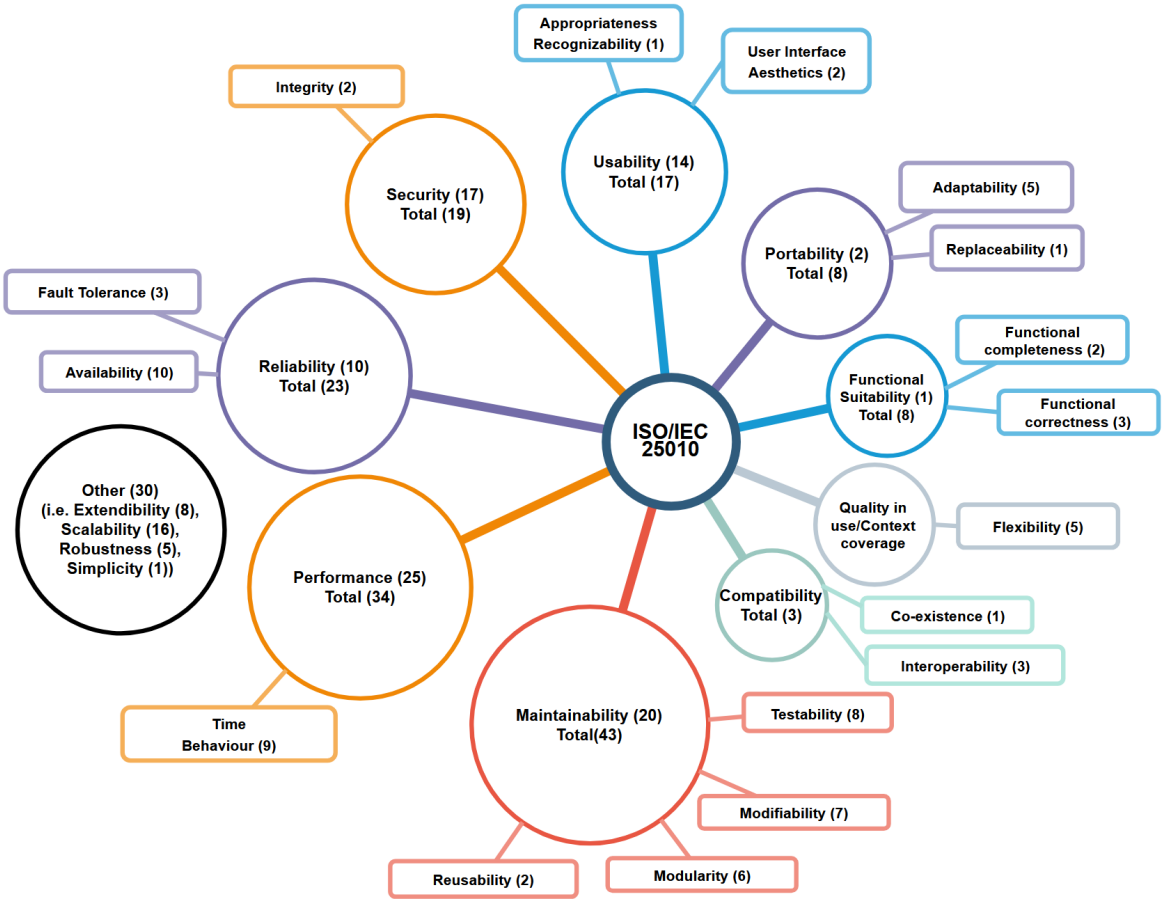


Figure 3.10 Frequency of quality attributes that the participants think they affect architectural decisions

We asked the participants if they seek out alternative options for their requirements, even when they already have a potential solution in mind during decision-making. Two individuals

Table 3.14 Correlations between the answers for Q9 (difficulties of making architectural decisions) and Q22 (factors that affect architectural decisions)

Factors	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	F15	F16	F17	F18	F19	F20	F21	Management preferences	Tool technology	Risk	Time	Quality attributes	User requirements	Maintenance	Personal experiences	Business goals	Previous decisions	Constraints
F2	0.513	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
F3	0.530	**0.013**	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
F4	**0.003**	0.867	**0.039**	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
F5	**0.015**	**0.038**	**0.004**	0.359	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
F6	**0.001**	0.212	0.216	**0.084**	**0.000**	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
F7	**0.002**	0.622	0.816	0.195	**0.000**	**0.000**	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
F8	**0.015**	0.661	0.618	**0.006**	**0.073**	**0.001**	0.185	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
F9	0.557	0.740	0.696	0.169	0.245	0.131	0.453	0.378	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
F10	0.108	0.373	0.113	**0.003**	0.957	0.132	**0.076**	**0.002**	**0.002**	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
F11	**0.090**	0.303	**0.001**	0.159	**0.021**	0.104	0.192	0.182	0.295	**0.007**	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
F12	**0.007**	0.741	0.373	0.305	**0.009**	**0.000**	**0.001**	**0.023**	0.592	0.934	**0.044**	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
F13	0.485	0.908	0.401	0.244	0.985	0.873	0.471	0.698	**0.044**	**0.029**	**0.095**	0.176	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
F14	**0.088**	0.277	0.396	0.216	0.514	**0.036**	0.707	0.304	0.217	**0.006**	**0.003**	0.269	**0.000**	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
F15	0.507	0.270	0.892	0.144	0.540	0.338	0.329	0.889	0.340	0.711	0.384	0.618	**0.000**	0.327	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
F16	**0.037**	**0.007**	0.120	0.377	0.535	0.143	0.034	0.787	**0.090**	0.974	**0.002**	0.828	0.391	0.233	**0.069**	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
F17	**0.002**	**0.009**	**0.012**	0.520	0.001	0.007	0.015	0.237	**0.022**	0.105	**0.079**	0.265	0.344	**0.092**	0.608	**0.001**	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
F18	0.167	**0.043**	0.178	0.106	0.712	**0.064**	0.923	0.525	0.977	**0.075**	0.321	0.720	**0.060**	**0.003**	**0.069**	**0.089**	0.533	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
F19	0.280	0.179	**0.051**	0.538	0.183	0.613	0.569	0.359	0.593	0.661	0.670	**0.082**	0.151	0.379	0.736	0.335	0.468	**0.092**	...	...	...	...	...	...	...	...	...	...	...	...	...	...
F20	0.738	0.562	**0.067**	0.380	0.427	0.886	0.466	0.393	0.658	0.897	0.828	0.506	0.117	0.378	0.898	**0.039**	0.618	0.437	**0.000**	...	...	...	...	...	...	...	...	...	...	...	...	...
F21	**0.005**	0.254	**0.078**	0.131	0.866	0.104	0.298	0.966	**0.047**	**0.012**	**0.000**	0.443	0.167	**0.038**	0.270	**0.000**	**0.008**	**0.009**	0.295	**0.018**	...	...	...	...	...	...	...	...	...	...	...	...
Management preferences	0.446	0.828	0.260	0.725	**0.012**	0.535	**0.004**	0.835	**0.055**	0.276	0.146	0.621	**0.036**	0.193	**0.089**	0.292	**0.012**	0.456	0.240	0.848	0.128	...	...	...	...	...	...	...	...	...	...	...
Tool technology	0.456	0.800	0.124	0.119	0.967	0.789	0.579	0.832	0.311	**0.033**	0.100	0.285	0.229	**0.039**	0.725	0.586	0.432	0.888	0.279	0.311	0.355	0.175	...	...	...	...	...	...	...	...	...	...
Risk	0.228	0.167	**0.084**	0.771	0.628	**0.033**	0.968	0.573	0.126	0.816	0.800	0.928	0.734	0.532	0.921	0.227	0.646	**0.012**	0.146	**0.014**	0.664	0.724	0.763	...	...	...	...	...	...	...	...	...
Time	0.205	**0.018**	0.149	0.151	**0.021**	**0.046**	**0.011**	0.130	0.810	0.232	0.420	0.589	0.896	0.106	**0.032**	**0.002**	**0.019**	**0.008**	0.943	0.515	0.764	**0.022**	0.623	**0.005**	...	...	...	...	...	...	...	...
Quality attributes	0.526	0.306	0.243	0.352	0.970	0.187	0.826	0.867	0.129	0.712	0.663	**0.015**	0.738	0.486	0.944	0.704	0.127	0.793	0.231	0.277	0.805	0.142	0.193	**0.000**	0.580	...	...	...	...	...	...	...
User requirements	0.319	0.210	**0.096**	0.596	**0.027**	**0.022**	0.578	0.791	0.231	0.981	0.274	0.589	0.556	0.308	0.334	0.618	0.328	0.396	0.415	0.494	0.340	0.552	0.343	0.243	0.368	**0.003**	...	...	...	...	...	...
Maintenance	0.478	0.333	0.090	0.257	0.826	0.929	0.434	0.875	0.901	0.209	0.435	0.504	0.268	0.321	0.473	0.514	0.219	0.465	**0.097**	0.031	0.759	0.995	0.620	0.124	0.113	**0.001**	**0.001**	...	...	...	...	...
Personal experiences	0.247	0.916	**0.010**	0.491	0.427	**0.029**	0.866	**0.090**	0.838	**0.050**	0.881	0.207	0.149	0.931	0.785	**0.086**	**0.033**	0.104	0.445	0.486	0.802	0.694	0.288	0.689	0.746	**0.058**	0.521	0.201	...	...	...	...
Business goals	0.161	0.833	0.952	0.979	0.585	0.344	**0.094**	0.187	0.654	0.565	0.254	0.202	0.530	0.708	0.124	0.111	0.944	0.814	0.228	0.223	0.862	0.133	0.422	**0.011**	0.951	**0.062**	**0.009**	**0.002**	0.791	...	...	...
Previous decisions	**0.030**	0.194	0.784	**0.054**	0.428	0.701	0.078	0.565	0.232	0.980	0.591	0.743	**0.013**	**0.056**	0.211	0.884	0.140	**0.025**	0.603	0.345	0.439	0.791	**0.005**	0.655	0.919	0.372	0.635	0.202	0.448	0.749	...	...
Constraints	0.142	**0.079**	0.503	0.204	**0.017**	0.058	0.186	0.439	0.746	0.558	0.705	0.315	0.167	0.180	0.834	**0.096**	**0.023**	0.364	0.367	0.352	0.133	0.615	0.181	0.899	**0.072**	0.621	0.269	0.969	0.286	0.221	0.004	...
Costs	0.840	**0.070**	0.666	0.974	**0.001**	**0.053**	**0.001**	0.860	0.621	0.724	0.248	0.384	0.626	**0.089**	**0.050**	**0.082**	**0.002**	0.514	0.626	0.547	0.351	**0.005**	0.205	0.630	**0.000**	0.929	0.959	0.297	0.382	0.462	0.761	**0.000**

Note: p values obtained with the Spearman's rank-order correlation. p values less than the 0.1, 0.05 and 0.01 significance level are shown in bold.

did not respond to this query. Of the 99 respondents, the majority, 90 individuals, confirmed they do look for alternatives, while 7 indicated they do not. One person noted they do so "in some cases, we implement the predicted architecture based on experience". A considerable number of respondents revealed that their selection from among the alternatives is typically influenced by their past experiences. Other prevalent methods mentioned include conducting a tradeoff analysis, engaging in discussions with stakeholders, and evaluating the advantages and disadvantages. A selection of the responses provided by the participants is as follows.

- P17: *"It evolves through our straw-man/brainstorming process with stakeholders."*
- P39: *"Based on the information available and the experience of the group of decision-makers."*
- P54: *"Tradeoff between the alternatives. Chose the one that meets the business need for the least cost and risk."*
- P89: *"By considering cost, complexity and the popularity of the architecture."*

We asked the participants on the validation of their architectural decisions and the methodologies they employ for such verification. Out of 65 of the participants who responded, 26 indicated they do not confirm their architectural decisions, and 8 mentioned the absence of a formal method for verification. From the remaining 31 respondents, varied methods were cited including ATAM (N:15) [77], CBAM (N:3) [78], Proof of Concept (PoC) (N:4) [79], alongside Review and Discussions with stakeholders. ATAM was noted as the most frequently used method among the mentioned techniques.

Building on the findings of Razavian et al. [80], which highlighted the need for improvements in software architecture decision-making, we formulated a question specifically related to this area. Examples of these deficiencies include the need for lightweight techniques or tools to guide decision-making, making the decision-making more agile and more efficient information sharing.Participants were presented with certain areas highlighted in boxes in Figure 11 and were asked to identify which they believed required

changes. The counts of the participants' choices for these areas are displayed at the base of each box in Figure 3.11. The responses revealed a consensus over half of the participants pointing out the need for better information sharing and tracking of decisions and their underpinnings as crucial for improving decision-making.

Figure 3.11 Areas to be improved for better architectural decisions

3.4. Threats to Validity

In our investigation, we aimed to gather data on prevalent methods in the practice of architectural decision-making, including the technical and societal obstacles encountered and the various factors influencing these decisions. We disseminated a survey among a broad group of 101 software architects from 27 different countries, striving to garner a representative set of responses. Despite our efforts, there are potential limitations and validity risks that we intend to discuss.

Construct validity: pertains to the degree to which a tool accurately measures what it's supposed to [81]. We solicited and incorporated input from domain experts to align our survey with established theoretical frameworks, selecting these experts from participants in a prior exploratory study. Based on their insights, we refined and expanded our questions,

particularly about quality aspects. We designed conditional questions to precisely reflect our research aims, such as only showing certain questions based on previous answers, thereby enhancing the survey's clarity and relevance. These adjustments, guided by expert feedback, were aimed at reducing any threats to our survey's construct validity, including refining the questions to eliminate any ambiguities and clearly define our measurement goals. Although we grounded our survey in solid, tested concepts, the potential reluctance of participants to share candid views about their workplaces was identified as a risk, which we mitigated by ensuring anonymity and confidentiality. Additionally, our method of transforming textual responses into categorized data was carefully managed to prevent loss of detail.

Internal validity: focuses on the relationship between independent and dependent variables and the correctness of the inferred causal links [81]. We corroborated our findings with existing literature to diminish the threat to internal validity and applied a logical cause-and-effect analysis where it was appropriate. The diversity of our participants, spanning 27 countries, posed a potential for cultural or regional bias, which we aimed to mitigate through random selection.

In order to strengthen internal consistency and ensure the reliability of the instrument used in our study, we conducted an assessment using Cronbach's Alpha, as detailed in Section 3.2.2.3.. The analysis revealed results indicating satisfactory internal consistency between the various sections of our questionnaire and underlined the methodological rigor used in the development and validation of the questionnaire. Such a systematic approach to assessing and confirming the reliability of the questionnaire significantly supports the integrity of our research findings.

External validity: examines the extendibility of the study's results to other contexts, populations, or conditions [81]. We endeavored to enhance the representativeness of our findings by enlarging our sample size, thereby attempting to ensure the findings' broader applicability. Our sample included diverse participants, varying in their organizational roles, levels of hierarchy, geographical locations, and project scopes, which we believed would support the generalizability of our conclusions.

A significant constraint of our research was the predominance of respondents with limited experience in the field, which could introduce a bias. We addressed this by including questions that less experienced respondents could also answer. The demographic homogeneity of the respondents, particularly in terms of nationality and gender, was noted but not expected to affect the overall applicability of our findings, based on prior interviews.

3.5. Evaluation and Discussion

In this section, we focused on the results of our study to identify the challenges encountered in the architectural decision-making process, factors affecting the architectural decision-making process, and possible improvements that can be made in the architectural decision-making process. The objective of our research was to delve deeply into the prevailing practices of decision-making concerning software architecture within the industry. Following a detailed analysis based on individual questions, we've extracted responses to our research queries. A summary of the key insights, in relation to the research questions specified in Section 3.2., is presented in Table 3.15. This table highlights the new discoveries of our research in bold font, while at the same time listing the results that support the findings from previous studies in the far right column.

Table 3.15 Summary of findings

RQs	Main Findings	Similar Findings
RQ.1., RQ.1.1, RQ.1.2, RQ.1.3	**From the responses, we understood that unstructured approaches are preferred in the architectural decision-making process. Decisions are made by teams of about two or three people, usually through discussion-based approaches, such as brainstorming and consensus.** Preferring unstructured approaches that do not require systematic participation may result in biased information shared within the group. Documenting decisions is a way to reduce biases. **While there is no specific format for decision documents, alternatives and tradeoffs are the most common areas used for documentation.** Generally, decision documents are stored in web-based collaboration software such as Confluence. To track decisions that affect each other, document reviewing is a commonly used approach.	[53]
RQ.2.	**We observed that costs, quality attributes, and user requirements are the most influential factors for architectural decisions. When the ISO/IEC 25010 quality factors are considered, maintainability, performance, and reliability are found to be the most influential factors in decision making. The challenging situations with the highest average agreement among participants are large business impact (F14), and dependencies with other decisions (F13).**	[4], [52], [82]

Continued on next page

Table 3.15 – continued from previous page

RQs	Main Findings	Similar Findings
RQ.2.1	A Mann-Whitney U test was conducted to test the hypotheses under this RQ. **The findings show us that there is a difference between those who make the decisions by one person and those who make the decisions with more than one person in terms of the F1, F5, F12, F13, and F17 situations. These situations are related to stakeholders, which supports the conclusion. F1, and F12 are significant for both architecture decision documentation and the group size of the decision-making group.**	
RQ.2.2.	A Mann-Whitney U test was conducted to test the hypotheses under this RQ. **Our results show that there was no significant effect for any of the factors examined with respect to the number of decision-makers, whether decisions are documented or not and whether agile lifecycle is used or not.**	
RQ.3.	**We understood that the architectural decision among different alternatives is usually made intuitively and based on experience.** This may not be a problem when the decision-maker has years of experience in a particular problem domain. Otherwise, this may lead to poor decisions. **According to responses, final decisions are generally not verified by any formal method.**	
RQ.4.	**We noticed during the study that information sharing is mentioned many times by participants in different questions. This shows that information sharing has an important role in the decision-making process. According to the results, efficient information sharing is also mentioned as an area that needs improvement. Brainstorming sessions, informal discussions, and meetings are some of the activities that contribute to information sharing at different levels.**	

The feedback from our study's participants, based on their professional experiences, indicates that information sharing plays a crucial role throughout various phases of the architectural decision-making process. Yet, there lacks a uniform system for architectural documentation to facilitate this communication effectively. This gap often leads those involved in decision-making to rely on their intuition rather than a defined methodology. Our research reveals that both the dissemination of information and the maintenance and monitoring of made decisions tend to be unstructured. Typically, due to the high level of effort required, tracking decisions becomes challenging, especially when relying on meeting minutes rather than thorough documentation. This situation underscores the necessity for further research into developing documentation practices that can efficiently transmit information with minimal effort. Existing templates for recording architectural decisions do exist, such as the Tyree/Akerman template [37], the principal decisions template from Bredemeyer Consulting [83], and M. Nygard's architectural decision records (ADR) [84] in his blog post. It is advisable for companies to adopt such standardized templates to streamline their decision-making processes.

The ambiguous connection between requirements and the documentation of software architecture often leads to a lack of traceability within practical applications. Traceability of architectural decisions within the software development lifecycle offers numerous benefits. It enhances the comprehensibility of software, aids in analyzing the impact of changes, and assists in the assessment of design [31]. Given that many architectural decisions recur across projects, the accumulated knowledge from prior decisions, successful or otherwise, can become invaluable, especially if it's organized systematically. We inquired with our participants if they monitor interrelated architectural decisions within their projects or organizations. A majority, 78 respondents, confirmed they do track these decisions, typically through reviews of documentation and meetings. Conversely, 24 participants acknowledged they do not track decisions. Those who do follow the decisions noted the process is manually intensive and lacks efficiency. This highlights a gap and suggests a demand for methodologies or tools that would allow architects to perform follow-ups with less effort and in a structured manner.

The survey data makes it clear that numerous elements influence the decision-making process. Architects are required to weigh a multitude of factors simultaneously, balance the potential gains and losses, and then make informed decisions. This complexity runs the risk of certain factors being neglected or incorrect decisions being made. The situation underscores a pressing need for methodologies or instruments that can aid architects in making decisions more efficiently and methodically. In this vein, the development of varied approaches and tools could provide significant support to architects in their decision-making endeavors.

3.5.1. Making and Documenting Decisions

Drawing from participant feedback, it became evident that a substantial number of them favor recording at least some architectural decisions and storing such records on web-based platforms like Confluence [85]. This underscores the significance of documentation within project development.

These records are generally accessible to all developers involved in a project, facilitating more straightforward and efficient information exchange among stakeholders. The predominant method for tracking interrelated decisions, as revealed by responses to the query "how the decisions that affect each other are tracked?", is through document reviews. This preference for document reviews underscores the value placed on documentation.

From the insights gathered on "which decisions are documented?", it's clear that decisions concerning "services, tools and technologies" and those at the "system-level" are typically documented, aside from those respondents who indicate that they document all types of decisions. Responses to question 21, "What types of architectural decisions do you think are typically more critical to the project in practice?", highlighted that decisions shaping subsystems, components, layers, and those pertaining to technology are deemed critical for practical projects. Furthermore, documented decisions are also considered pivotal in practice, as per the responses to question 21.

The content within decision documents is varied, as indicated by the responses. While some documents may not be comprehensive, others include extensive details like the decision-maker, decision drivers, and associated risks. Analysis revealed that alternatives, tradeoffs, and diagrams are the most commonly included elements in decision documents.

Responses also indicated that decision-making is typically a team effort, with teams averaging about five members. Our findings suggest that team-based decision-making is advantageous for a deeper understanding of issues. Decisions made unilaterally may not be fully grasped by those not involved in the decision-making process, leaving them unaware of the decision's rationale, importance, or relevance. However, when decisions are made collectively, all members gain a thorough understanding of the issues and the reasoning behind the decisions.

Various methods are employed in group decision-making. Reviewing the selection frequency of each group decision-making approach offered to participants, brainstorming emerged as the most favored method. Additionally, multiple methods were often cited in responses. The joint application of brainstorming and consensus was noted 35% of the time, illustrating their common usage. These approaches encourage open-mindedness and free expression of ideas without the fear of judgment, signifying that diverse perspectives from all project developers are valued.

The importance of differing viewpoints and the sharing of knowledge is further emphasized in the responses to question 28 ("Which of the following areas could be improved to make better architectural decisions? Do you have any additional areas to recommend?"). This highlights the collective consensus on the value of varied insights in the decision-making process.

3.5.2. Influence and Compelling Factors

The answers to the 9^{th} question ("In the light of your experience in making architectural decisions in past projects, could you evaluate the influence of the following difficulties as 5

(very influential), 4 (influential), 3 (neutral), 2 (not so much influential), 1 (not influential at all)?") shed light on the hurdles encountered by participants in decision-making. When faced with making decisions, participants identified "The decision had a major business impact" as the most formidable challenge, in contrast, "The decision had too few alternatives" was considered the least challenging. This contrasts with our prior exploratory study [21], where "F14 - The decision had a major business impact" ranked as the third most challenging, and "F9 - The decision had too few alternatives" remained the least challenging. Over half of the respondents viewed all types of architectural decisions as critical, with the exception of those related to organization and tools. These findings suggest a multitude of factors are influential and crucial in decision-making processes. Participants predominantly recognized costs as significantly influential, followed by quality attributes, user needs, and limitations. The prominence given to cost and constraints resonates with Tang et. al.'s [82] research on design rationale. However, the impact of management preferences was not deemed as prominent. This is consistent with findings from Miesbauer et. al. [54]—a semi-structured exploratory study involving 9 experts across 6 companies—which revealed that the impact of previous decisions was perceived as substantial. Regarding quality aspects affecting architectural decisions, performance was indicated as most impactful, with maintainability and security trailing closely. From these insights, it can be inferred that striking a careful balance between performance and cost is pivotal in architectural decision-making. Respondents emphasized that while performance is critical for quality attributes, the cost is a major driver in shaping architectural decisions.

3.5.3. Final Decision and Validation

Decision-making often involves a large number of alternatives, and the act of selecting from these alternatives constitutes a significant segment of the decision-making process. Responses gathered from our survey indicate that a large number of participants actively seek out alternative solutions, even when a potential solution is already conceived. While relying on their experience is commonly cited when navigating among alternatives, the process of evaluating which alternatives best fulfill their needs and making an informed

choice by weighing the tradeoffs was also mentioned as a prevalent practice. In addition, 34 respondents disclosed that they do not employ any formal method to validate their ultimate decision. Among those who do perform verifications, the ATAM [77] and PoC [79] methods are the favored techniques.

3.5.4. Improvements

In pursuit of addressing our fourth research question, we posed the question to our participants: "which areas need improvement in the decision-making process so that better decisions can be made?". The majority highlighted the need for enhancements in the domains of information dissemination and the documentation of decisions and their underlying justifications. As previously noted, information exchange plays a pivotal role at every phase of decision-making. It was observed that architects frequently cited various communication methods like discussions, meetings, and emails when responding to inquiries about their architectural decision-making practices. Consequently, it is unsurprising that effective information sharing was the most recurrent theme in response to question 28 ("Which of the following areas could be improved to make better architectural decisions? Do you have any additional areas to recommend?"), posed in the concluding part of our survey. Gaps in information and suboptimal sharing practices can introduce uncertainties in the decision-making process [86]. Such challenges could be mitigated by fostering the exchange of diverse viewpoints and knowledge among those making the decisions, thereby diminishing uncertainty levels.

4. SOFAR-DSS: An Advanced Decision Support System for Architectural Design Patterns Using OpenAI and DBpedia

Software architecture serves as the basic blueprint of a software system by shaping the interaction and functionality of its components. It is crucial in defining the quality characteristics of the system, such as performance and reliability, and ensuring that the software remains adaptable and efficient in the dynamic environment of technology. It is crucial to address the intricacies of architectural decision making because these decisions carry significant weight in the software lifecycle. During the decision-making process, decision-makers encounter a variety of challenges. These can stem from an abundance of alternatives, a lack of necessary information, or prevalent uncertainties. Such challenges can obscure the path to optimal decisions, making it difficult to evaluate the trade-offs associated with each option. In light of these complexities, there is a clear need for a tool that simplifies the decision-making process. This tool would ideally aid in distilling the array of choices, illuminate areas obscured by insufficient data, and provide clarity amid the often-ambiguous nature of technological evolution, thereby guiding architects toward decisions that align with both current requirements and future adaptability. We have developed a tool that addresses RQ.5. stated in Section 1.1. SOFAR-DSS, a sophisticated tool designed to streamline the decision-making process for software architects, is an AI-powered tool that simplifies complex decision-making processes by appealing not only to experienced architects but also to a wider audience involved in the architectural design process [22]. It provides ease of use with a user-friendly interface. SOFAR-DSS is notable for its AI algorithms and interactive design, which provide accurate, user-centered guidance. This helps reduce the cognitive load on users and enhances the quality of architectural decisions.

In this section, we examine the journey from the development to the application of SOFAR-DSS, showcasing its capabilities and functionalities as a marker of innovation in the field of software architecture. We examine its intelligent query processing mechanism,

its integration with DBpedia's rich repository, and the practical advantages these features bring to the architectural process. We will also present the results of rigorous evaluations with expert software architects, highlighting the practical effectiveness of the system and its potential to revolutionize architectural decision-making in software engineering. Through this discussion, we aim to underline the transformative impact of SOFAR-DSS on the field and lay the groundwork for a deeper analysis of its role in shaping the future of software architecture.

4.1. Related Work

In this section, we aim to review and critically evaluate the work and methods prevalent in this area, specifically examining the role of Decision Support Systems (DSS) in facilitating the complex process of making architectural decisions in software engineering. Existing work has largely focused on two major methodologies for architecture evaluation: ATAM [77] and CBAM [10]. These methodologies have been fundamental in informing and shaping existing approaches to software architecture decision making. ATAM emphasizes the evaluation of various software quality attributes, such as reliability and performance, and provides a systematic way to guide trade-offs and improve communication between stakeholders. It reveals the impact of different architectural decisions on software quality attributes and promotes clearer communication between stakeholders, including maintainers, users, customers and developers, by refining and refactoring requirements. Furthermore, ATAM lends itself to continuous design and analysis efforts. Despite these strengths, ATAM has shortcomings, particularly in the management of uncertainties and comprehensive analysis of quantifiable quality metrics such as response time and latency. Limitations in methodological support tools also create difficulties in its practical implementation. In contrast, CBAM builds on the principles of ATAM by integrating economic considerations by evaluating the cost-effectiveness of various architectural options. It supports decision making by quantifying the economic trade-offs of different architectural options in light of system quality objectives such as performance, availability and security. Using scenarios based on quality attribute responses, CBAM quantifies the value of different architectural

solutions. However, CBAM's primary focus on cost-benefit analysis may not fully capture all stakeholder objectives, especially with respect to certain software quality attributes such as response time and latency. CBAM uses specific formulas and assigned benefit points to determine the utility of an architecture, which can sometimes reduce the complexity of the interaction between scenarios to an oversimplified quantitative measure that abstractly assesses importance and likelihood.

Farshidi and colleagues [87] developed a DSS explicitly targeting Pattern-Driven Architecture, designed to simplify the process for software architects when selecting the most appropriate architectural patterns. This system processes the functional and quality requirements entered by architects and suggests a set of patterns that are closely aligned with these needs. Architects have the option to modify the initial requirements to fine-tune the system's recommendations to determine the most appropriate patterns for their specific project. This DSS is based on fundamental software engineering approaches, including ISO/IEC software quality models and the MoSCoW prioritization method. Architects can identify their main needs, such as high availability or accessibility for an application, which the system uses to drive the pattern selection process.

Within the realm of aiding software engineers in decision-making, delving into multi-criteria decision-making (MCDM) methods appears to be a fruitful direction [88]. Svahnberg et al. [11] introduce a quantitative approach that simplifies the comparison of possible architectures via the Analytical Hierarchy Process (AHP) [89]. This method systematically collects stakeholder preferences for particular quality attributes and lends a hand in forming a quantifiable perspective on the pros and cons of various architectural alternatives. It establishes a framework underpinned by a multi-criteria decision method, which allows for the juxtaposition of different software architecture candidates against distinct quality attributes. However, this method might not scale well for extensive, complex projects where a multitude of interconnected design decisions are at play. Advancing the work of Svahnberg et al. [11], Rita et al. [90] have enhanced the field of MCDM methods with their Hybrid Assessment Method (HAM). Acknowledging the shortcomings of AHP, such as its semi-linear scale and the extensive need for pairwise comparisons, HAM was designed to

be a simpler yet more effective method in aiding software development decisions. It aims to assess the impact of various decisions on software projects, striking a balance between criteria weights that are compensatory, thereby sidestepping some of the known limitations related to scaling and ranking methods.

Over a four-year span, Capilla and associates [91–96],have developed a web-based system for managing Architectural Design Decisions (ADDs). This platform offers capabilities for documenting, exploring, and visualizing ADDs, and it enables the collaborative management of these decisions. The ADDs platform is noted for its ability to trace and depict the evolution of architectural knowledge (AK) through an iterative process, mirroring the way software architects gradually refine architectures. It accommodates assigning statuses and categories to decisions aligned with different phases of the software lifecycle, like development and maintenance, thereby enriching the relevance of the decision's context.

A range of tools for handling architectural knowledge, like ADDS, are available. The Archium tool, brought forth by Jansen and Bosch in 2005 [2], ensures continuity of knowledge across a spectrum of concepts from inception through a system's entire lifecycle. Archium is known for its encompassing approach, dealing with everything from initial requirements to final implementation details. Another tool, AREL, by Tang and colleagues [97], aids architects in formulating and documenting architectural design, placing a strong emphasis on the rationale behind architectural decisions and outcomes. It meticulously documents various aspects of architectural knowledge, including concerns, decisions, and outcomes. The Knowledge Architect suite, developed by Jansen and others [98] in 2008, is an extensive set of tools dedicated to the capture, management, and dissemination of architectural knowledge, featuring a repository and server that store a wide array of knowledge entities and several clients for handling this knowledge in various forms and contexts.

Question Answering (QA) is a defined linguistic task designed to sift through vast data to deliver precise information in response to user queries [99]. In software engineering, QA models serve multiple roles. The MSRbot by Abdellatif and team [100], for example,

leverages data from sources like Git and Jira to address software project inquiries in plain language. Calle Gallego and others [101] devised QUare, a QA model specifically for eliciting requirements, which includes a meta-ontology to aid in question generation and initial structuring for software domains. Additionally, Lian and colleagues [102] created an ontology realignment to organize and store abstracted requirements, enabling the identification of potential requirement statements by matching document passages with domain-specific ontological queries.

Yet, our research indicates a shortfall in the application of QA models in software architecture decision-making. To bridge this gap, SOFAR-DSS integrates a QA model into this process, marking a departure from models that are typically focused on requirements elicitation or broad query resolution. Our system, customized for the specific demands of software architecture, employs advanced AI and semantic analysis to propose architectural pattern recommendations that are pertinent to the given context. This distinctive feature accentuates the innovative nature of our approach, imbuing the software architecture decision-making process with AI-powered insights. SOFAR-DSS thus becomes a crucial tool for architects and decision-makers by providing custom recommendations that consider specific project needs and the context at hand.

In contrast to previously mentioned methodologies and tools that concentrate on managing and evaluating architectural knowledge, SOFAR-DSS adopts an active stance, interpreting user inputs and recommending design patterns using AI and the semantic web, such as DBpedia. This unique feature of SOFAR-DSS is its interactive QA model that engages with users to grasp their specific requirements and context, differing from traditional, more static methods dependent on preset criteria and documentation. This dynamic characteristic of SOFAR-DSS allows for instantaneous pattern recommendations, adapting to the fluctuating demands of software projects, which represents a notable advancement from established methods like ATAM and CBAM, or the repository-centric approaches like Archium and Knowledge Architect. Ultimately, SOFAR-DSS stands out for its interactive, AI-assisted approach to architectural decision-making, remedying the shortcomings of existing systems by adapting to the diverse and immediate needs of software development.

4.2. Methodology

In this section, we unveil SOFAR-DSS, a cutting-edge tool that integrates a QA model with the DBpedia ontology to propose specific design patterns for software architecture. The primary aim of this innovative tool is to offer custom design pattern recommendations that address the unique issues identified by software architects or decision makers. This innovative process capitalizes on the QA model's advanced ability to process and understand detailed architectural inquiries and utilizes the extensive, organized knowledge base provided by the DBpedia ontology. The functionality of the proposed system starts with the analysis of architects' queries through the QA model, aligns these queries with the detailed semantic framework of the DBpedia ontology and accordingly identifies appropriate software architecture design patterns. The goal is to provide recommendations that are not only technically precise and context-sensitive, but also adaptive to the dynamic environment of software architecture.

Positioned at the intersection of natural language processing (NLP), knowledge representation techniques and software engineering principles, this methodology aims to simplify complex decision-making processes in software architecture. It provides an intelligent system that can comprehend a wide range of architectural requirements and propose feasible, informed solutions. Our goal is to streamline the decision-making process, alleviate the cognitive burden on architects or decision makers, and promote the selection of optimal architectural designs informed by the latest, comprehensive industry insights.

Our approach incorporates the analytical strengths of OpenAI's gpt-3.5-turbo-instruct [103] engine for generating initial recommendations, which are further validated through DBpedia's extensive database. This dual strategy ensures that our suggestions are both innovative, leveraging state-of-the-art AI technology, and credible, underpinned by DBpedia's semantic depth. The combination of AI-driven creativity with semantic web-based validation forms a solid foundation for informed, effective decision-making in software design. The operations are performed sequentially from left to right, as illustrated in Figure 4.1, our methodology's workflow is detailed, beginning with data extraction

from the organization's JIRA system, a leading platform for project management and issue tracking. Through the JIRA REST API, our system programmatically gathers issue data, facilitated by a Python interface that connects with the JIRA API, allowing the DSS to access up-to-the-minute project information and user-reported problems. In this context, Atlassian JIRA Software Cloud [104]is exemplified as a model for incorporating project management and issue tracking systems within our DSS framework, although other systems can be similarly integrated. The data collected from JIRA undergoes analysis to determine its relevance to design, with design-related issue entities being identified via DBpedia Spotlight. Subsequent steps involve generating recommendations through the QA model and enriching these recommendations with DBpedia, offering users a comprehensive set of suggestions.

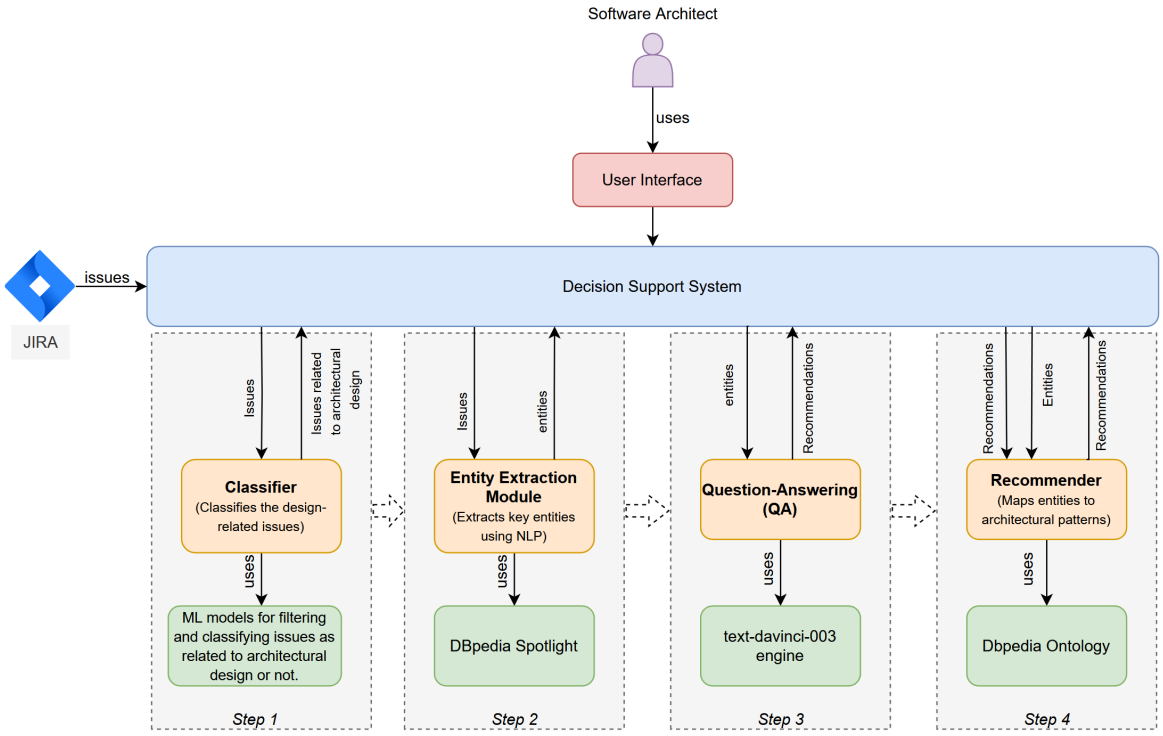


Figure 4.1 Model Diagram for SOFAR-DSS

4.2.1. Step 1: Classifier Module

This section delves into the specifics of the dataset employed within the classification module, elaborating on the various classification methodologies implemented, the details

of these approaches, and the outcomes derived from their application.

4.2.1.1. Dataset

For the classification module, which serves as the initial phase of our proposed model, we utilized a dataset created by Bhat et al. [28]. This dataset derives from two prominent open-source software (OSS) projects: Apache Spark and Apache Hadoop. The study involved extracting issues related to these projects from JIRA. Two software architects, each with over five years of experience, independently analyzed the issues. They classified these issues as either pertaining to design decisions or not related to design decisions. In total, 2,139 issues were examined for decision detection, with 781 issues identified as design decisions and 1,358 categorized as unrelated to design decisions. The architects manually segmented the dataset into two distinct classes: ”Design Decision” and ”Not a Design Decision,” employing specific criteria outlined in Figure 4.2 below. Following the established guidelines, the architects conducted a manual review of the text within the summary and description fields of all extracted issues. Issues lacking a description or whose purpose could not be deduced from the text provided were designated as deleted. Those issues that fit into a specific decision-making category were appropriately labeled and identified as a Design Decision. Conversely, issues that did not align with any decision-making categories were classified as Not A Design Decision.

Following the data extraction and manual labeling steps conducted by Bhat and colleagues, we further analyzed the dataset with our specific needs in mind. During this reevaluation, we identified and removed issues that contained insufficient information. As a result, our refined dataset consists of 1,519 issues, of which 728 are classified as not related to design, and 791 are deemed related to design.

Figure 4.3 illustrates the comprehensive workflow of our text classification pipeline, detailing a methodical process for analyzing and categorizing textual information. The initial phase, ’Data Preprocessing,’ involves refining the raw text data by removing any extraneous elements and standardizing its format. This step includes breaking down the text into

Structural decision:

+ Adding or updating plugins, libraries, or third-party systems
+ Adding or updating classes, modules, or files (a class, in this context, refers to a Java class)
+ Changing access specifier of a class
+ Merging or splitting classes or modules
+ Moving parts of the code or the entire files from one location to another (code refactoring to address maintainability issues)
+ Updating names of classes, methods, or modules

Behavioral decision:

+ Adding or updating functionality (methods/functions) and process flows
+ Providing configuration options for managing the behavior of the system
+ Adding or updating application programming interfaces (APIs)
+ Adding or updating dependencies between methods
+ Deprecating or disabling specific functionality
+ Changing the access specifiers of methods

Ban decision:

+ Removing existing plugins, libraries, or third-party systems
+ Discarding classes, modules, code snippets, or files
+ Deleting methods, APIs, process flows, or dependencies between methods
+ Removing deprecated methods

Design decision:

+ An issue that belongs to any one of the above categories

Not a design decision:

+ An issue that does not belong to any of the above categories

Figure 4.2 Rules for manual classification. (Source: Bhat, Manoj and Shumaiev, Klym et al.,"Automatic extraction of design decisions from issue management systems: a machine learning based approach",ECSA 2017)

manageable pieces (tokenization) and normalizing these elements to maintain consistency across the dataset. Next, the 'Feature Extraction' phase uses as feature extraction techniques the Term Frequency-Inverse Document Frequency (TF-IDF) and Bag of Words (BoW). TF-IDF assess the significance of each word within a document relative to a collection of documents, serving as a crucial factor in understanding text relevance [105]. The BoW

model It helps to go from text to numbers by counting the occurrence of words within a document, and machine learning algorithms work with this numerical data. [106].

To ensure the classification model's effectiveness and generality, we implement a k-fold cross-validation technique, setting k to 10. This method divides the dataset into k subsets, using each in turn for validation while the remainder serves as training data. This cycle helps in evaluating the model's stability and accuracy across different segments of the data, offering a comprehensive understanding of its predictive capabilities. This method provided a multi-faceted validation, mitigating the risk of overfitting and allowing for a comprehensive assessment of the models' generalization capabilities.

In the 'Model Training' stage, we select and refine a machine learning algorithm tailored to the specific characteristics of our text data and the classification objectives. During training, the algorithm adjusts its internal parameters to align with the patterns derived from the TF-IDF and BoW features and their corresponding labels, learning to predict accurately.

Following training, the 'Model Testing' phase critically assesses the trained model using a separate set of data not seen during training. This evaluation is essential for verifying the model's ability to generalize its predictions to new, unseen data, ensuring its applicability and reliability in practical settings.

Finally, the 'Results' section presents a detailed analysis of the classification results, including metrics such as accuracy, precision, recall, and others. These indicators provide insights into the model's performance, highlighting its strengths and areas for improvement in text classification tasks. Each stage of the pipeline plays a crucial role in achieving a refined, effective classification process that accurately interprets and categorizes textual data, aiming for optimal performance in real-world applications.

4.2.1.2. Classification Algorithms and Model Selection

In the classification section of our study, we employed a comprehensive approach to text classification by utilizing both traditional machine learning algorithms and advanced

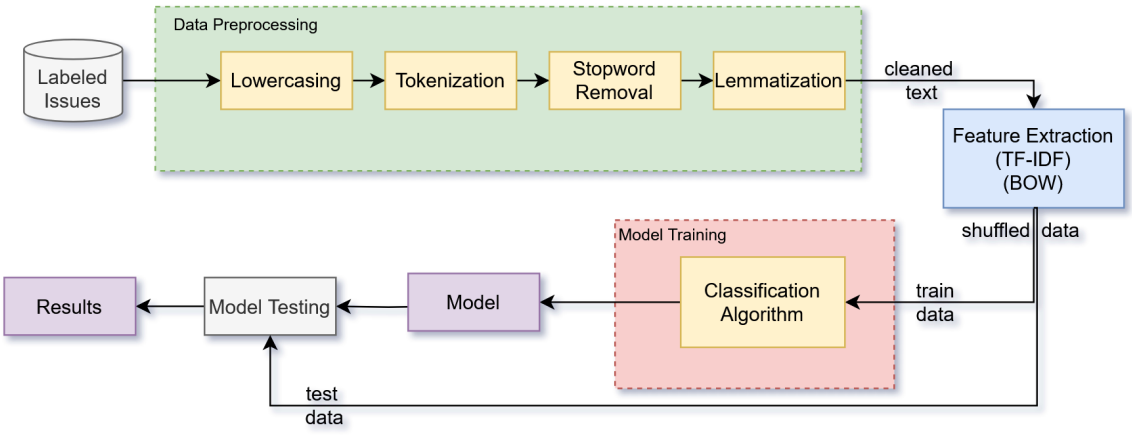


Figure 4.3 Text classification pipeline employed in SOFAR-DSS

deep learning techniques, specifically BERT (Bidirectional Encoder Representations from Transformers) [107] and LSTM (Long Short-Term Memory) [108]. To ensure the robustness and reliability of our classification models, we adopted the k-fold cross-validation method throughout our experiments. This section details the performance outcomes of each algorithm, reflecting their effectiveness in the context of our text classification objectives.

Traditional Machine Learning Algorithms

We began our analysis with a selection of traditional machine learning algorithms, known for their efficacy in handling structured data and offering interpretable results. The algorithms tested include Random Forest (RF) [109], Support Vector Machine (SVM) [110], k-Nearest Neighbors (kNN) [111], Extreme Gradient Boosting (XGBoost) [112] and Light Gradient Boosting Machine (LGBM) [113]. Each algorithm was subjected to k-fold cross-validation, with k set to 10, to mitigate any bias or variance in the model's performance and ensure a comprehensive evaluation across different data segments.

In our comprehensive analysis, we explored two prominent techniques for text vectorization: TF-IDF and BoW. These methods serve as critical preprocessing steps, transforming raw text into structured, numerical formats that machine learning algorithms can process effectively. TF-IDF emphasizes the importance of terms based on their frequency across documents,

distinguishing them by their uniqueness. In contrast, the BoW approach simplifies text representation, focusing solely on the occurrence of words within a document, disregarding grammar and word order but capturing the essence of the text's vocabulary. By applying both TF-IDF and BoW methodologies, we aimed to capture a broad spectrum of features from the text data, assessing how each vectorization strategy influences the performance of the selected machine learning models in our text classification tasks. This dual approach allowed us to draw more nuanced conclusions about the effectiveness of each algorithm under different data representation schemes, enriching our understanding of their practical applications in text analysis.

In our study, meticulous optimization of hyperparameters was essential to ascertain the most effective settings for the algorithms deployed. The objective was to enhance each model's accuracy and its capability to generalize across various datasets. For the purpose of fine-tuning these hyperparameters, we employed the GridSearchCV technique, a robust tool provided by the Scikit-Learn [114] library in Python, renowned for its efficiency in machine learning tasks. This technique exhaustively explores a myriad of parameter combinations outlined in a predefined grid, evaluating each to pinpoint the configuration that delivers the optimal performance based on a predetermined evaluation criterion. Leveraging the insights garnered from GridSearchCV, we meticulously trained our models with these optimally identified parameters. The specifics of these parameters, which were instrumental in achieving superior model performance, are detailed in Table 4.1, illustrating the rigorous approach adopted to ensure the models' effectiveness and reliability in prediction tasks.

Table 4.1 Best parameters of each algorithm used in classification

Algorithm	**Best parameters**
LGBM	"colsample_bytree": 1, "learning_rate": 0.02, "n_estimators": 400
RF	"n_estimators": 150, "min_samples_split": 20, "min_samples_leaf": 4, "max_features": "sqrt", "max_depth": 80, "bootstrap": False
XGBoost	"learning_rate": 0.1, 'max_depth': 8, "n_estimators": 200
KNN	"knn_leaf_size": 2, "knn_n_neighbors": 42, "knn_weights": "distance"
SVM	"SvmVM gamma": 60, "pca_n_components": 4

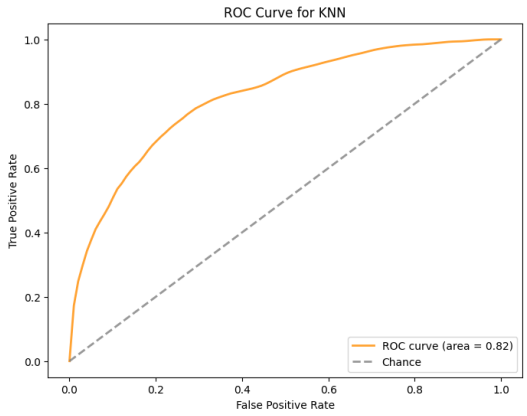
A comprehensive evaluation was conducted on the performance of these machine learning algorithms applied to text classification tasks. The models were rigorously assessed based on their accuracy, recall, precision, and F1-score metrics for both training and testing datasets, ensuring a thorough examination of their predictive capabilities. The results of each algorithm is given in 4.2.

Table 4.2 Comparison of Machine Learning Models using TF-IDF and BoW

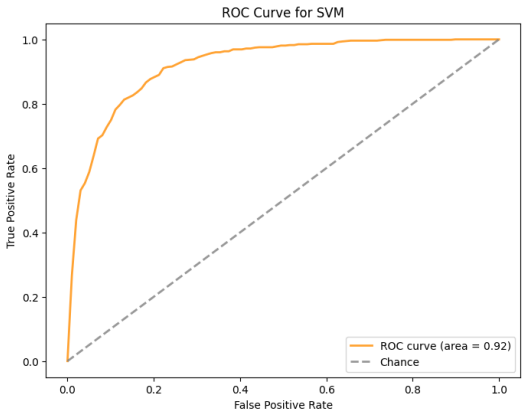
	Accuracy		Recall		Precision		F1	
	Train	Test	Train	Test	Train	Test	Train	Test
				TF-IDF				
KNN	0.768	0.734	0.764	0.737	0.772	0.738	0.765	0.730
SVM	0.985	0.847	0.985	0.847	0.985	0.847	0.985	0.846
RF	0.890	0.817	0.891	0.819	0.891	0.820	0.890	0.817
LGBM	0.929	0.807	0.929	0.806	0.929	0.807	0.928	0.804
XGBoost	0.911	0.798	0.908	0.798	0.912	0.799	0.911	0.796
				BoW				
KNN	0.682	0.63	0.671	0.621	0.756	0.699	0.648	0.584
SVM	0.933	0.833	0.934	0.832	0.933	0.834	0.933	0.832
RF	0.872	0.823	0.873	0.825	0.874	0.825	0.872	0.822
LGBM	0.908	0.809	0.907	0.808	0.908	0.809	0.907	0.807
XGBoost	0.909	0.801	0.909	0.801	0.909	0.802	0.909	0.798

Each algorithm's performance was further scrutinized through Receiver Operating Characteristic (ROC) curves, with the Area Under the Curve (AUC) serving as a critical indicator of their classification efficacy. The ROC AUC graphs offer a visual and quantitative understanding of each model's true positive rate versus false positive rate, providing an intuitive comparison of their performance. The results, presented in tabulated form for clarity, are complemented by these graphical representations to deliver a holistic view of the models' operational effectiveness in handling text-based data, thereby framing a solid foundation for the subsequent discussions and conclusions drawn in this study. The ROC

curves for each algorithm are given in the graphs below. Figure 4.4, Figure 4.5, and Figure 4.6 for the cases where TF-IDF vectorization is used. Figure Figure 4.7, Figure 4.8, and Figure 4.9 are the plots when BoW is used.

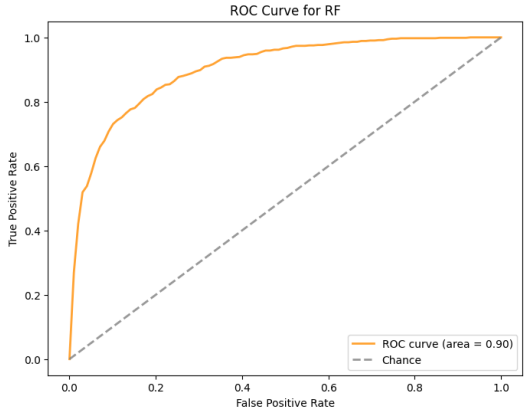


(a) ROC Curve for KNN

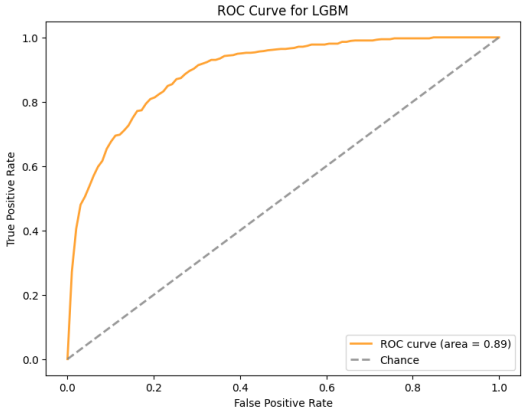


(b) ROC Curve for SVM

Figure 4.4 ROC Curves for KNN and SVM

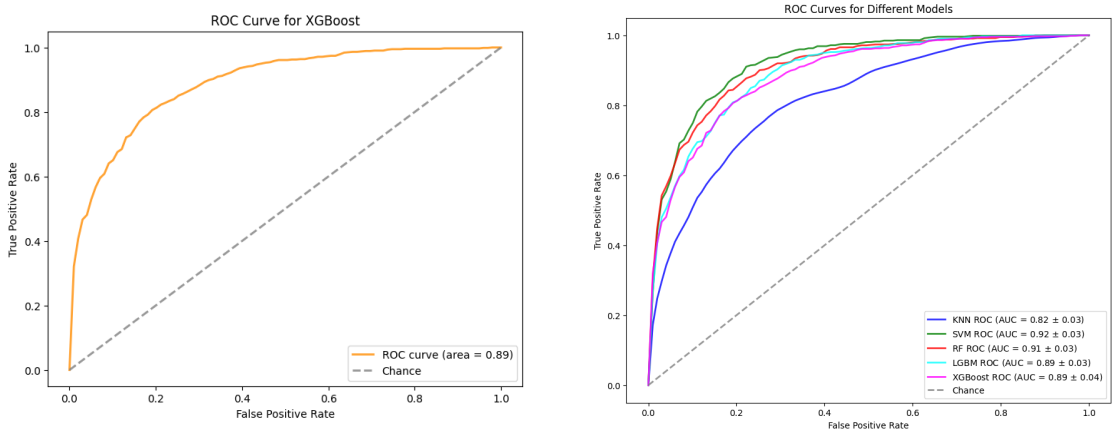


(a) ROC Curve for RF



(b) ROC Curve for LGBM

Figure 4.5 ROC Curves for RF and LGBM

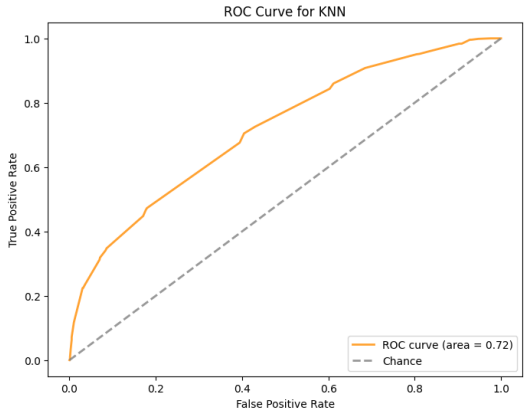


(a) ROC Curve for XGBoost

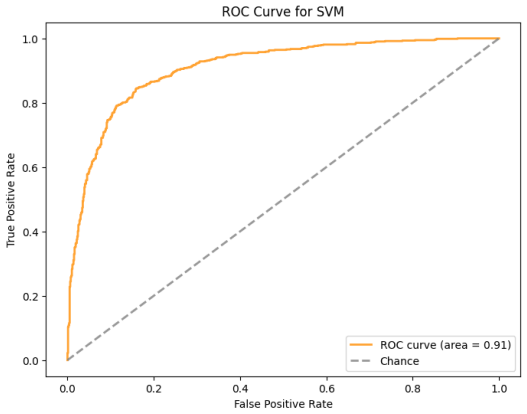
(b) ROC Curve for all algorithms

Figure 4.6 ROC Curves for XGBoost and all algorithm comparisons

In our analysis, focusing on ROC AUC graphs generated post-classification with Term Frequency-Inverse Document Frequency (TF-IDF) vectorization, its evident that all models exceed the performance of random prediction, as delineated by the 'Chance' line. Overall, examining all graphs, while all models outperform random prediction (as indicated by the 'Chance' line), SVM stands out with the highest AUC, followed closely by RF, LGBM and XGBoost, with KNN lagging slightly behind. This comparative result is also shown in Figure 4.6(b). These differences in AUC reflect the different degrees of trade-offs that each model makes between sensitivity (true positive rate) and specificity (1 - false positive rate). The more closely the curve follows the left boundary and then the upper boundary of the ROC space, the more accurate the test. Conversely, the closer the curve follows the 45-degree diagonal of the ROC space, the less accurate the test.

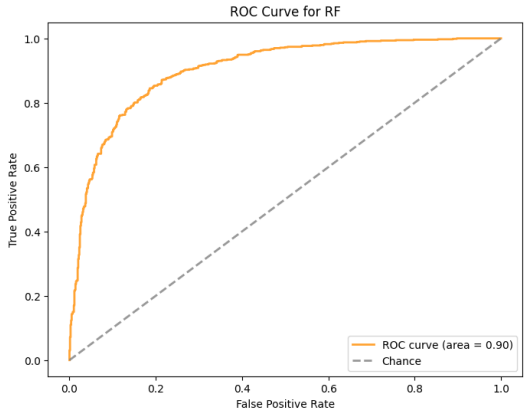


(a) ROC Curve for KNN

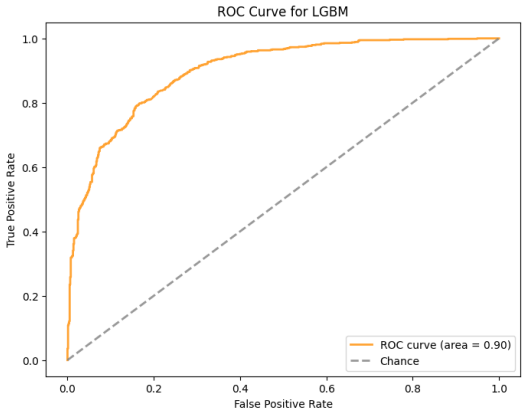


(b) ROC Curve for SVM

Figure 4.7 ROC Curves for KNN and SVM (BoW)

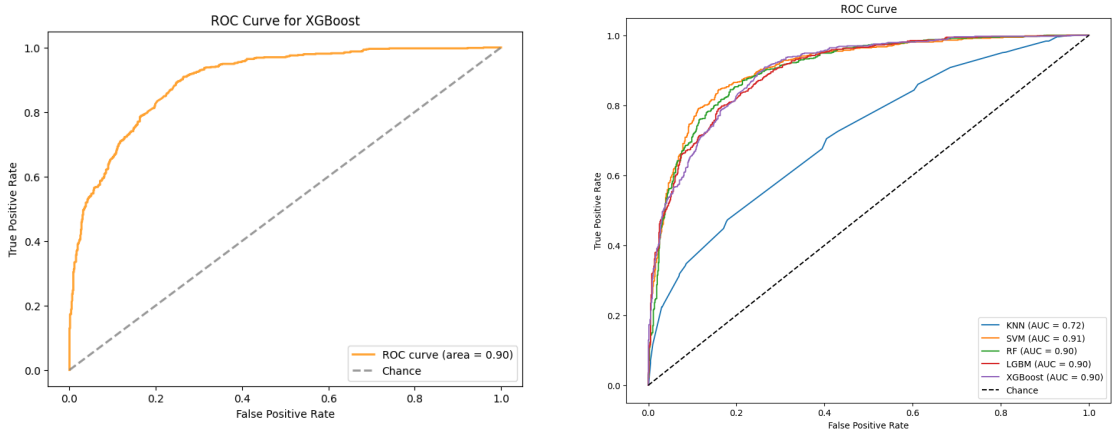


(a) ROC Curve for RF



(b) ROC Curve for LGBM

Figure 4.8 ROC Curves for RF and LGBM (BoW)



(a) ROC Curve for XGBoost

(b) ROC Curve for all algorithms

Figure 4.9 ROC Curves for XGBoost and all algorithm comparisons (BoW)

The ROC curves also displayed above provide a visual comparison of the performance of various classification algorithms when applied to a dataset processed using the BoW technique. The curve for SVM, with an Area Under Curve (AUC) of 0.91, suggests a high level of distinguishability, indicating that this model has a strong capability to differentiate between classes. The RF and LGBM models show comparable performance with an AUC of 0.90, which implies that they are also highly effective, though slightly less so than SVM. The XGBoost model exhibits an AUC of 0.90 as well, suggesting a robust performance. Meanwhile, the KNN model, with an AUC of 0.72, demonstrates a fair classification capability, but it lags behind the other models.

The 'Chance' line, depicted by the dashed diagonal line, represents a random guess scenario. All models clearly outperform this baseline, with the SVM model's curve hugging the top left corner the most, reflecting its superior accuracy. The consistent positioning of the curves above the 'Chance' line across all algorithms corroborates their effectiveness when utilizing the BoW approach for data representation in the classification task.

Deep Learning Techniques

To leverage the capabilities of deep learning in understanding the nuances of natural language, we further explored text classification using BERT and LSTM models. These

models excel in capturing the contextual relationships within text data, making them particularly suited for tasks requiring a nuanced understanding of language.

Similar to the traditional algorithms, we applied k-fold cross-validation for these deep learning models. This process was crucial in assessing the models' generalization capabilities across unseen data and ensuring that our findings were robust against overfitting.

The performance of BERT and LSTM models is also reported using accuracy, precision, recall, and F1-score. Given the complexity of these models and their sensitivity to training data, these metrics are instrumental in understanding how well the models can predict the correct classes while accounting for the intricacies of language.

BERT: Text classification, a pivotal task in natural language processing (NLP), involves assigning predefined categories to text. The emergence of Bidirectional Encoder Representations from Transformers (BERT), developed by Devlin et al. [107], has significantly advanced the state-of-the-art in this domain. BERT's architecture, leveraging the transformer model introduced by Vaswani et al. [115], employs a novel pre-training and fine-tuning methodology that has shown remarkable performance across a wide range of NLP tasks, including text classification.

BERT's innovation lies in its ability to understand the context of a word in a sentence more effectively than prior models. It does this by pre-training on a large corpus of text with two objectives: masked language modeling and next sentence prediction. This pre-training enables the model to grasp a deep understanding of language nuances and structures. For text classification specifically, BERT considers the entire context of a sentence or a sequence, allowing it to capture subtler meanings and relationships between words.

To apply BERT for text classification, the model is first pre-trained on a large text corpus, such as Wikipedia or the BooksCorpus ([116]), enabling it to learn a broad understanding of language. It is then fine-tuned on a specific text classification task with a smaller dataset. During fine-tuning, the final layer of BERT is adapted to the specific classification task, and the model learns task-specific nuances, significantly improving its predictive performance.

A substantial benefit of BERT is its transfer learning capability. Once pre-trained, BERT can be fine-tuned with relatively small datasets and still achieve high accuracy, making it highly efficient for tasks where labeled data is scarce. This has been demonstrated in various studies, such as Sun et al. [117], where BERT achieved state-of-the-art results on multiple text classification benchmarks with minimal task-specific adjustments.

However, the implementation of BERT is not without challenges. Its complexity and size demand considerable computational resources, particularly for training. This has led to the development of optimized versions like DistilBERT [118] and ALBERT [119], which maintain comparable performance while being more efficient.

BERT represents a significant leap forward in text classification, offering unparalleled accuracy by understanding the context of words more effectively. Its impact is evident across a range of applications, from sentiment analysis to topic classification, showcasing its versatility and power in harnessing the subtleties of language for NLP tasks.

In our thesis, we used the BERT (Bidirectional Encoder Representations from Transformers) model for text classification tasks. Leveraging the transformers library by Hugging Face, we establish a workflow for processing textual data, fine-tuning the BERT model, and assessing its efficacy on a dataset presumed to consist of issue texts with classification labels.

Our methodology begins with importing the dataset from a CSV file and preparing the textual and label data for processing. We employed the BertTokenizer from the 'bert-base-uncased' pretrained model to tokenize the texts, adapting them into a format the model can understand. This step includes appending special tokens, implementing padding, and generating attention masks to help the model differentiate between actual content and padding.

Following tokenization, we convert the processed texts into a TensorDataset, which simplifies data manipulation and batching during the training phase. We use Stratified K Fold cross-validation for dividing the dataset into training and testing subsets.

For each fold, we establish DataLoader instances for both the training and testing datasets, facilitating batch processing and shuffling of the training data to enhance the model's

generalization capabilities. We fine-tune a BERT model, configured similarly to the 'bert-base-uncased' pretrained model but adapted for binary classification (num_labels=2), with the AdamW optimizer and a linear rate scheduler for learning. The training process involves forward and backward propagation, where the model learns from the provided batches of input ids, attention masks, and labels. The best parameters identified for BERT in our tasks include a batch size of 32, a learning rate of 2e-5, and training over 4 epochs.

Upon completing the training, we evaluate the model's performance on both the training and testing sets, calculating metrics such as accuracy, precision, recall, and F1 score. These metrics quantitatively assess the model's ability to accurately classify new, unseen texts.

LSTM: Long Short-Term Memory (LSTM) networks, a special kind of Recurrent Neural Networks (RNNs), are designed to overcome the limitations of traditional RNNs in capturing long-term dependencies in sequence data. Introduced by Hochreiter and Schmidhuber in 1997 [108], LSTMs have been pivotal in advancing the field of natural language processing (NLP), particularly in tasks requiring the understanding of context over long sequences, such as text classification.

The key innovation of LSTMs lies in their architecture, which includes memory cells that store information for long periods and gates that regulate the flow of information into and out of these cells. This design allows LSTMs to remember and forget information selectively, making them highly effective for processing and making predictions based on long sequences of data, such as sentences or documents in text classification tasks.

In text classification, LSTMs analyze the input text sequentially, preserving information from earlier in the sequence to influence the understanding and classification of the content that comes later. This capability is crucial for accurately capturing the meaning of a text, as the context provided by preceding words can significantly influence the interpretation of the subsequent ones. For instance, LSTMs have been applied successfully in sentiment analysis, where the sentiment conveyed by a sentence can depend heavily on the sequence in which the words appear.

Several studies have demonstrated the effectiveness of LSTMs in text classification. For example, Sutskever et al. [120] showed that LSTMs could generate coherent text sequences and perform well in language modeling tasks. Similarly, Tang et al. [121] utilized LSTMs for sentiment analysis and opinion mining, achieving state-of-the-art results on multiple datasets. These successes underscore the model's proficiency in handling the nuances of language, making it a popular choice for various NLP applications.

Despite their strengths, LSTMs also come with challenges. They are computationally intensive and can be slow to train, especially on large datasets. Additionally, designing and tuning LSTM networks to achieve optimal performance can require significant expertise and experimentation. To address these issues, researchers have proposed various optimizations and alternatives, such as Gated Recurrent Units (GRUs) and attention mechanisms, which can offer similar benefits with less complexity.

LSTMs represent a significant advancement in the ability to process sequential data, particularly text. Their design enables them to capture long-term dependencies that are crucial for understanding and classifying text. As research in the field continues, the methodologies around LSTMs and their applications in text classification are expected to evolve, further enhancing their utility and effectiveness.

We also used Long Short-Term Memory (LSTM) network for text classification tasks, leveraging the PyTorch framework for deep learning. Initially, the dataset is loaded and preprocessed, involving the removal of missing values, extraction of texts and labels, and conversion of labels to numerical format. The texts are then tokenized using BERT's tokenizer, padded to uniform length, and encoded into tensors alongside attention masks, preparing the data for the LSTM model. The defined LSTM classifier includes an embedding layer for text representation, a dropout layer to mitigate overfitting, and a fully connected layer for classification output. Training and evaluation functions facilitate the model's learning through cross-entropy loss optimization and performance assessment via accuracy, precision, recall, and F1 score metrics. Stratified K-Fold Cross-Validation is applied to ensure comprehensive model evaluation across diverse dataset subsets. The model undergoes

multiple training epochs, adjusting its parameters to improve text classification accuracy. The model uses the Adam optimizer and includes a dropout of 0.3 to prevent overfitting. The optimal training parameters are a batch size of 32, a learning rate of 0.001, and a longer training period of 15 epochs.

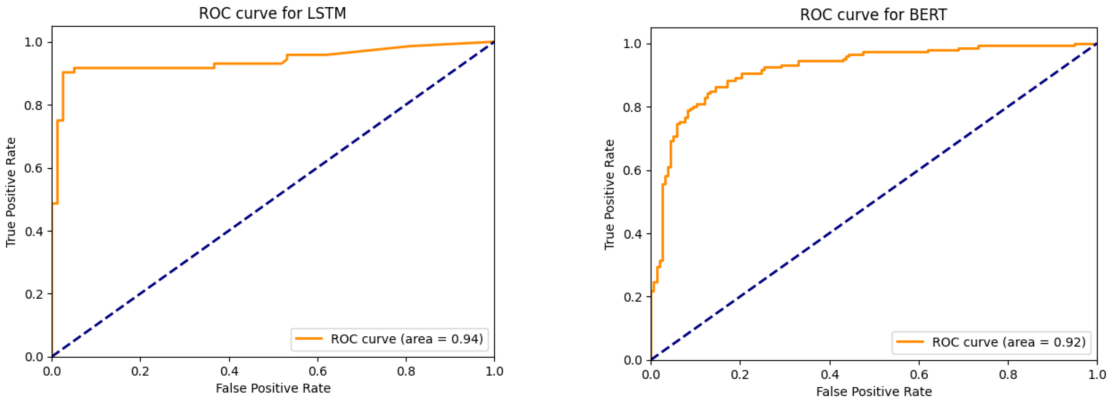
For each algorithm, we present the following metrics: accuracy, precision, recall, and F1-score. These metrics provide a multi-faceted view of each model's performance, allowing us to assess not only its overall correctness but also its ability to balance false positives and negatives effectively.

Table 4.3 Performance comparison of LSTM and BERT models

Model	**Accuracy**		**Precision**		**Recall**		**F1**	
	Train	**Test**	**Train**	**Test**	**Train**	**Test**	**Train**	**Test**
LSTM	0.89	0.86	0.89	0.86	0.89	0.86	0.89	0.86
BERT	0.9	0.84	0.9	0.84	0.9	0.84	0.9	0.84

The ROC curves also displayed in Figure 4.9(a) and Figure 4.9(b) for LSTM and bert algorithm respectively to provide a visual comparison of the performance of algorithms. The first graph shows the ROC curve for an LSTM model. The area under this ROC curve (AUC) is 0.94, indicating a very high level of diagnostic ability. The closer the AUC is to 1, the better the model is at distinguishing between the positive class and the negative class. An AUC of 0.94 suggests that the LSTM model has a strong ability to discriminate between the classes with a high true positive rate even as the false positive rate remains relatively low.

The second graph represents the ROC curve for a BERT model. Similar to the LSTM graph, it shows the model's True Positive Rate (TPR) versus False Positive Rate (FPR). The AUC for the BERT model is 0.92, which is slightly lower than the LSTM's but still indicates a high diagnostic ability. Both models exhibit excellent performance, but the LSTM model has a slight edge over the BERT model in this specific case.



(a) ROC Curve for LSTM

(b) ROC Curve for BERT

Figure 4.10 ROC Curves for LSTM and BERT algorithms

Upon establishing the superior performance of a specific classification algorithm over others through comparative analysis, our next step was to delve into the intricacies of the classification mechanism. This was achieved by integrating explainable artificial intelligence (XAI) techniques, specifically LIME (Local Interpretable Model-agnostic Explanations) [122] and SHAP (SHapley Additive exPlanations) [123], into our study. The application of these XAI methods aimed to shed light on the decision-making process of the chosen algorithm, thereby increasing the transparency and comprehensibility of its classification actions. LIME and SHAP stand out in the realm of machine learning for their ability to clarify the operations of otherwise opaque models [124], effectively addressing the often encountered trade-off between model performance and its interpretability. This is particularly relevant when dealing with 'black-box' approaches, such as those found in deep learning [125] and various ensemble techniques [126–128], where understanding the model's reasoning process is challenging. Our analysis focused on deconstructing the model's decision-making for individual instances, offering a detailed exposition on how each classification was derived. This granular examination is thoroughly documented and discussed in Section 4.4.3., presenting an in-depth exploration of the model's interpretative processes and the implications of its decisions on a case-by-case basis.

Comparative Analysis

Following the individual performance evaluation of various machine learning algorithms in our text classification framework, we performed a comparative analysis including traditional algorithms as well as advanced deep learning techniques such as BERT and LSTM. This comprehensive comparison identifies the strengths and weaknesses of each approach, providing a nuanced understanding of their applicability in our text classification context.

Our empirical results reveal that while traditional machine learning models such as Naive Bayes, SVM and Random Forests provide a strong foundation for performance, more sophisticated deep learning models generally perform better. In particular, the LSTM model performed best in our evaluations, demonstrating its ability to capture temporal dependencies and nuances in language that are often missed by more traditional models. This advantage is attributed to the architectural design of the LSTM, which enables it to effectively learn from the sequential nature of text data, thus improving its prediction capabilities.

Our findings also show that the BERT model provides a robust framework for text classification tasks. However, the LSTM model in particular has been successful in our case, probably due to its recurrent structure, which is well suited for the type of text data we analyze. LSTM's ability to model long-range dependencies within text sequences has been an important factor contributing to its strong performance, outperforming traditional algorithms.

In conclusion, while BERT and other deep learning models offer state-of-the-art capabilities, the choice of model should depend on the specific requirements of the task at hand. Factors such as the availability of computational resources, the need for interpretability and the complexity of the text data should be carefully considered. In cases where the capture of temporal linguistic features is crucial, LSTM models are particularly effective and can provide the best performance, as evidenced in our analysis.

4.2.2. Step 2: Entity Extraction Module

In the identification of design-centric issues, our methodology emphasizes the critical extraction of essential entities, terms, and concepts. These elements are integral to our analytical process as they form the basis for constructing precise queries. These queries are subsequently processed by our advanced QA model, which delves into the architectural nuances and context of the issues.

The extraction of entities is a pivotal procedure within our strategy, as it establishes the foundation for accurately recommending software architectural patterns. To carry out this extraction effectively, we leverage the sophisticated tool DBpedia Spotlight [129]. DBpedia Spotlight excels in its ability to annotate and recognize entities automatically, linking textual content to corresponding DBpedia resources. This tool enables us to forge a deeper connection between the issues extracted from our dataset and the vast, structured knowledge encapsulated within the DBpedia knowledge base. In order to entity extraction by dbpedia spotlight, we used Spacy DBpedia Spotlight package [130] in python. This toolkit functions as an Entity Recognizer and Linker through the utilization of DBpedia Spotlight. It is capable of being integrated into an existing spaCy Language object, or alternatively, it can establish a new one from a blank pipeline. This module connects SpaCy to DBpedia Spotlight, allowing for the seamless retrieval of DBpedia entities within your documents, whether by tapping into the public web service or by leveraging your private instance of DBpedia Spotlight. The document entities, or doc.ents, are enriched with comprehensive details of the entities including their URI, type, and more.

In our process, each issue text from our dataset is fed into DBpedia Spotlight, which meticulously identifies and associates key terms and phrases with their corresponding entities in DBpedia. To optimize the entity recognition process, we have set the tool's confidence parameter to 0.75, ensuring a judicious equilibrium between precision and comprehensive recall. This calibration helps in mitigating the potential for overlooking relevant entities while maintaining a high level of accuracy in entity recognition.

The integration of DBpedia Spotlight in our entity extraction phase allows us to establish a detailed and contextually relevant understanding of each design issue's architectural significance. These extracted entities are then utilized to formulate structured queries, which are adeptly evaluated by our QA model. This approach not only enriches our recommendations for software architecture patterns but also enhances the specificity and applicability of our suggestions, catering to the unique needs and contexts presented by each design issue.

4.2.3. Step 3: Generating Recommendations with QA Model

Our approach initiates with the careful construction of queries, which are subsequently inputted into a QA model adept in deciphering and scrutinizing architectural concepts and terminology. This model meticulously processes the queries, engaging with the DBpedia ontology—a comprehensive repository of diverse software architecture patterns. Through this engagement, the model aligns the recognized entities and their correlated issues with the relevant patterns within the knowledge base.

At the outset of our methodological process, we employ the sophisticated capabilities of OpenAI's gpt-3.5-turbo-instruct engine [38], which is tasked with generating design pattern recommendations in response to user-submitted issues. The interaction with the OpenAI model unfolds in a series of meticulously orchestrated steps:

- OpenAI API Key Configuration: We commence by integrating the OpenAI API key into our system, facilitating a secure and seamless interface with the OpenAI services.
- Prompt Creation: We meticulously craft a prompt for each user-reported issue, encapsulating the essence of the problem and soliciting the AI model for suitable design pattern recommendations. An example generated prompt: *"Considering the issue of* {*issue*} *and its related entities identified by DBpedia Spotlight: entities, what are the most appropriate design patterns to address this issue?"* {issue} will be replaced by the issue entered by the decision maker using the system.

- Engagement with OpenAI's Engine: The artfully formulated prompt is dispatched to the gpt-3.5-turbo-instruct engine using OpenAI's 'Completion.create' API method. This step activates the AI's analytical faculties to interpret the prompt and proffer suggestions, with the output tailored to deliver focused recommendations by curtailing the extent of token generation.
- Recommendation Extraction: OpenAI's response encompasses a spectrum of possible design patterns, from which we distill the most relevant pattern that aptly addresses the issue at hand.

In this process, the interplay between our query formulation, the QA model's analysis, and the interaction with OpenAI's sophisticated AI engine forms a robust foundation for delivering targeted and contextually appropriate software architecture pattern recommendations.

4.2.4. Step 4: Recommender

The method for improving suggestions made by a QA model consists of two main actions: validation and enrichment. These steps can be greatly strengthened by incorporating searches within an external database like DBpedia.

4.2.4.1. Validation with External Database DBpedia

Upon receiving suggestions from a QA model, its important to assess their accuracy and relevance. This assessment can be conducted through DBpedia searches. Such searches would confirm the presence of the suggested entities within the DBpedia repository and retrieve extra information. This supplementary information could encompass detailed explanations, pertinent links, and classification information that fits the context of the suggestion. This dual approach of utilizing recommendations from a QA model and corroborating them via DBpedia presents a comprehensive strategy for confirming the correctness, relevance, and thoroughness of the provided data.

- **SPARQL Query Construction:** We construct SPARQL queries that correspond with the patterns suggested by the OpenAI model, with the intent of locating these patterns within the DBpedia ontology.

- **Executing Queries on DBpedia:** We perform these queries at the DBpedia SPARQL endpoint, obtaining comprehensive entries that include abstracts and categorizations related to the proposed design patterns. Verification through DBpedia ensures that the suggestions are rooted in recognized principles of software design.

- **Synthesis of Results:** We analyzed the outcomes from the DBpedia queries to confirm the consistency between the AI-provided suggestions and the verified entries in DBpedia. This integrative process leads to the provision of recommendations that are informed by AI and corroborated by an authoritative knowledge base. To further tailor the relevance and accuracy of these recommendations, we apply specific filters. Results are refined based on designated 'dcterms:subject' categories found within DBpedia. This refinement guarantees that the final recommendations align with the query not only contextually or functionally but also fit within the targeted conceptual or thematic realms, such as particular architectural models, software development methodologies, or the broader spectrum of software architecture. The Sparql query we utilized is given below:

```
PREFIX dbc: <http://dbpedia.org/resource/Category:>
PREFIX dct: <http://purl.org/dc/terms/>
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT DISTINCT ?pattern ?description ?label WHERE {{
    ?pattern dbo:abstract ?description ;
        dct:subject ?subject.
    ?pattern rdfs:label ?label.
```

```
    FILTER (
        (CONTAINS(LCASE(?label), "{pattern}") ) &&
        (
      ?subject = dbc:Software_architecture ||
      ?subject = dbc:Architectural_pattern_\(computer_science\) ||
      ?subject = dbc:Software_design_patterns
        )
   )
    FILTER (LANG(?description) = 'en' && LANG(?label) = 'en')
}}
LIMIT 10
```

This SPARQL query is adapted to search DBpedia for detailed information about software design patterns specifically suggested by a question and answer (QA) system. The Pattern placeholder is dynamically modified by the QA system's suggestion, so that the query is directly related to the user's query or the architectural problem at hand. The query aims to retrieve patterns related to the fields of Software Architecture, Architectural Patterns in Software Engineering and Software Design Patterns. Thus, it is queried whether the patterns suggested by the QA model are present in verified databases such as dbpedia. How the query components are compatible with this goal is described below:

PREFIX: This section defines abbreviations for the URIs used in the query to simplify its syntax. For example, dbc: is set as a shorthand for http://dbpedia.org/resource/Category:, which is the base URI for DBpedia categories.

SELECT DISTINCT: This statement specifies that the query will return unique results based on the variables listed (?pattern, ?description, ?label). It aims to avoid duplicate entries in the output.

WHERE: This clause defines the pattern that DBpedia resources (?pattern) need to match to be included in the results. It specifies that the resources must have an abstract (?description)

and belong to one of the specified categories (?subject). Additionally, these resources must have a label (?label), which is the name or title of the design pattern.

FILTER: The FILTER functions are used to refine the search criteria:

- The first FILTER ensures that the label of the pattern (converted to lowercase using LCASE()) contains the search keyword specified by pattern. This makes the search case-insensitive and more flexible. Also it checks that the subject of the pattern falls under one of the three specified categories related to software design and architecture.
- The last FILTER condition ensures that both the description and the label of the patterns are in English ('en'), providing consistency in the language of the results.

LIMIT: This limits the number of results returned by the query to 10, making the output more manageable and focused on the most relevant entries.

Overall, this SPARQL query is crafted to explore DBpedia for specific software design patterns that match a given keyword, ensuring that the results are relevant to software architecture and design principles and are presented in English.

4.2.4.2. Enrichment

We enhanced the suggestions provided by SOFAR-DSS by verifying additional design patterns on DBpedia. To achieve this, we utilized entities identified through DBpedia Spotlight to formulate a SPARQL query. This query was executed to retrieve alternative design patterns from DBpedia's ontology, which were subsequently incorporated into the list of recommendations. The query is given below. The first part is of this query is prefix declarations that are shortcuts for the namespaces used in the query. The rest part of the query is as shown below:

SELECT DISTINCT ?pattern: This line is asking the database to return unique (DISTINCT) results for the variable ?pattern, which will represent design patterns.

WHERE Clause: This is where the criteria for selecting the data is specified.

- ?pattern rdf:type dbo:DesignPattern . is looking for resources that are of the type ”DesignPattern” according to the DBpedia Ontology.
- ?pattern rdfs:label ?label . selects the human-readable label of the design pattern.
- ?pattern dbo:abstract ?abstract . selects the abstract (a summary or description) of the design pattern.
- The next three lines with dct:subject filter patterns that are categorized under “Software Architecture,” “Architectural Pattern (Computer Science),” and “Software Design Pattern,” ensuring that only resources associated with these categories are considered.

FILTER: This function restricts the results based on the given conditions. CONTAINS(LCASE(?label), LCASE(“${entityToQuery}”)) ||CONTAINS(LCASE(?abstract), LCASE(“${entityToQuery}”)) is checking if the search term entityToQuery (which you should replace with your actual term) appears in either the label or the abstract of the design patterns. The LCASE function is used to convert both the label/abstract and the search term to lowercase, making the search case-insensitive.
LIMIT: This limits the number of results returned to a maximum of 10.

```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbr: <http://dbpedia.org/resource/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX dbc: <http://dbpedia.org/resource/Category:>

SELECT DISTINCT ?pattern WHERE {
  ?pattern rdf:type dbo:DesignPattern .
  ?pattern rdfs:label ?label .
  ?pattern dbo:abstract ?abstract .
  ?pattern dct:subject dbc:Software_architecture .
```

```
?pattern dct:subject dbc:Architectural_pattern_\
(computer_science\) .
?pattern dct:subject dbc:Software_design_pattern .
FILTER (CONTAINS(LCASE(?label), LCASE("${entityToQuery}")) ||
CONTAINS(LCASE(?abstract), LCASE("${entityToQuery}")))
}
LIMIT 10
```

4.3. USER INTERFACE DEVELOPMENT

The interface for users acts as the front end to our system for recommending architectural patterns, facilitating interaction between users and the foundational models and algorithms. The user interface (UI) was constructed utilizing Streamlit, a Python-based library that facilitates rapid web application development with reduced code complexity. This portion of the document explicates the UI's design principles, operational capabilities, and its developmental process.

Our approach to UI design prioritized straightforwardness and operational effectiveness, allowing software architects to seamlessly explore the system and access recommendations effortlessly. By adopting Streamlit's array of widgets and design configurations, we aimed to foster an environment characterized by its straightforward and user-friendly interface. There are four different pages in the interface; a home page (4.11) where general information about SOFAR-DSS is given, a classifier page (4.12) where the results of the different algorithms used for classification within the scope of the thesis are given, a JIRA data recommendations page (4.13) where the user gives issues from JIRA specific to the user's project as input to SOFAR-DSS, and a single issue recommendations page (4.15) where the user can see the results for a single issue. Key functionalities offered by the UI include are summarized below:

Issue Submission: The interface allows users to input only one issue or import multiple issues from tools like Jira. This action initiates the backend processes for classifying the issue and extracting relevant entities.

Visualization of Extracted Entities: If the issues entered are classified as design related, the system presents the identified entities and offers users the opportunity to evaluate them.

Pattern Recommendation: The system suggests design patterns to the user according to the issue or issues entered and presents their descriptions in dbpedia to the user with a drop-down menu.

Feedback Mechanism: The platform incorporates a feature for users to appraise the pertinence of the suggestions provided, thereby contributing to the continuous enhancement of the recommendation algorithm.

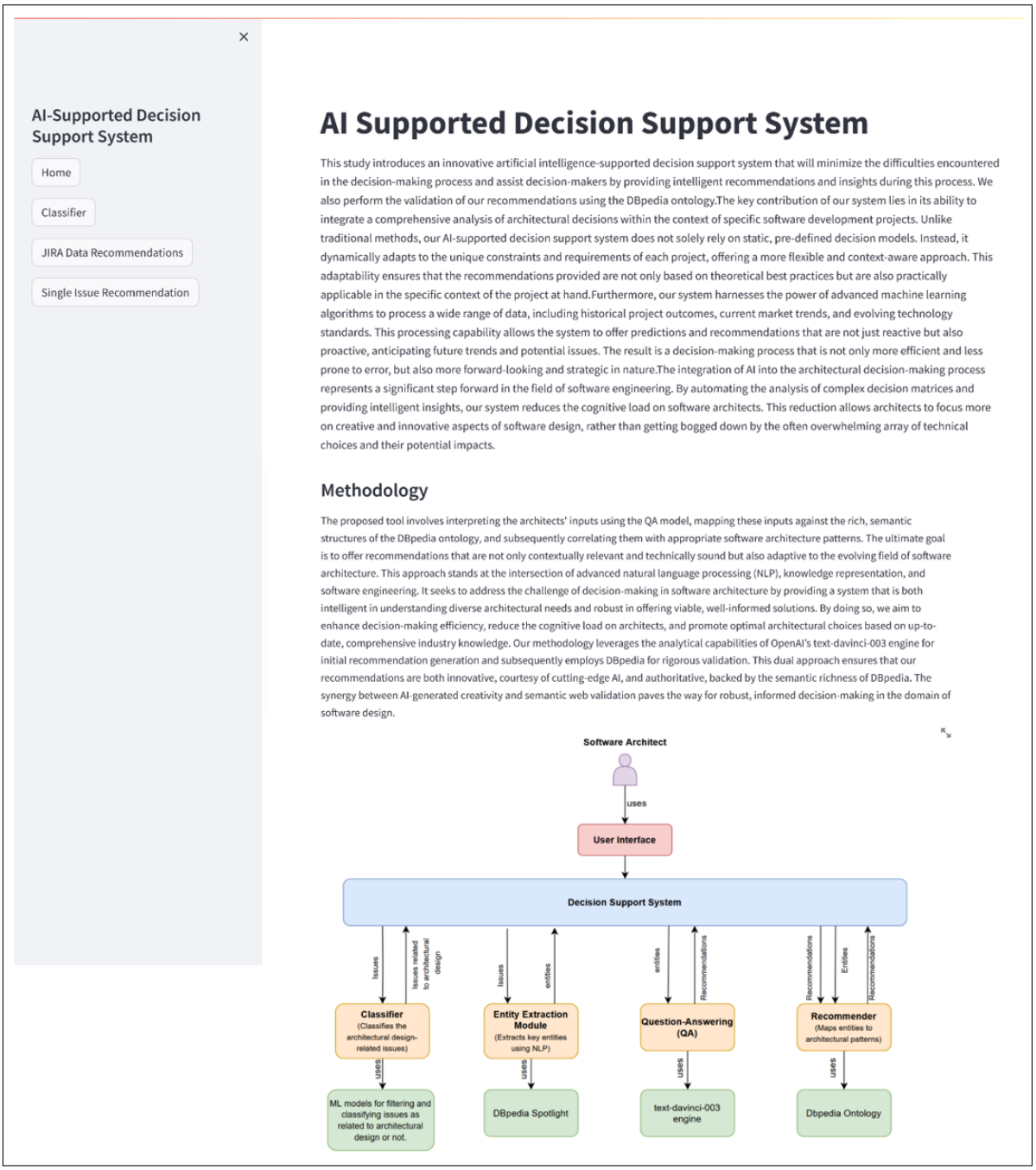


Figure 4.11 Home page of user interface

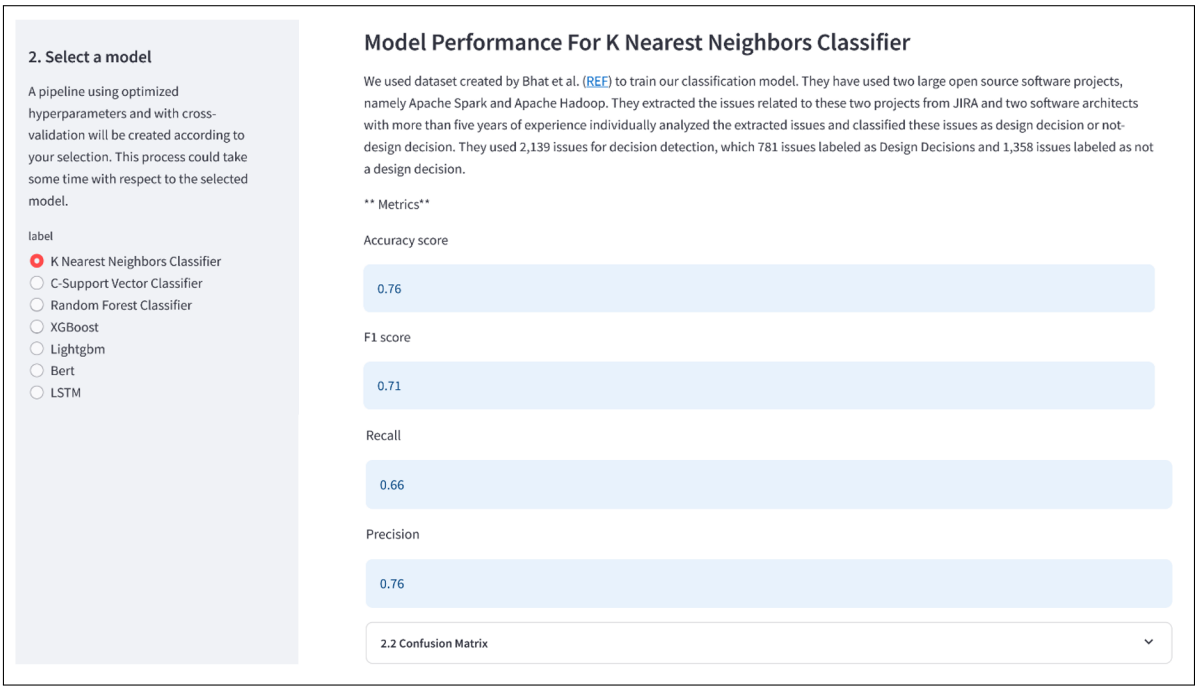


Figure 4.12 Classifier page of user interface

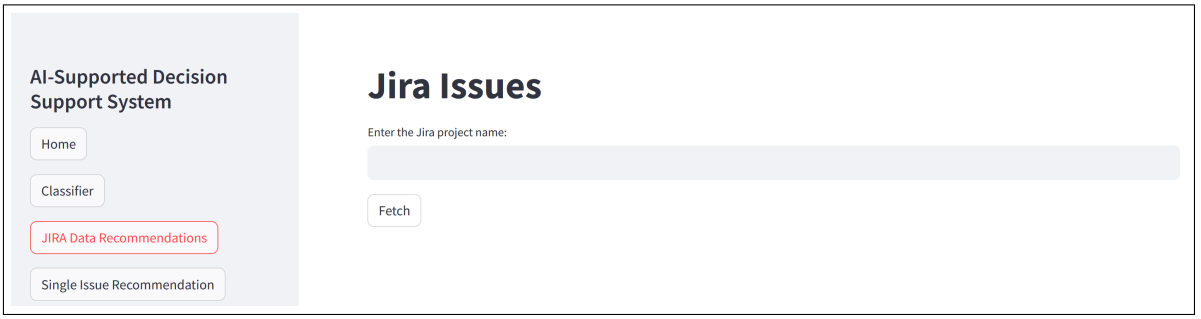


Figure 4.13 JIRA Data Recommendations page of user interface

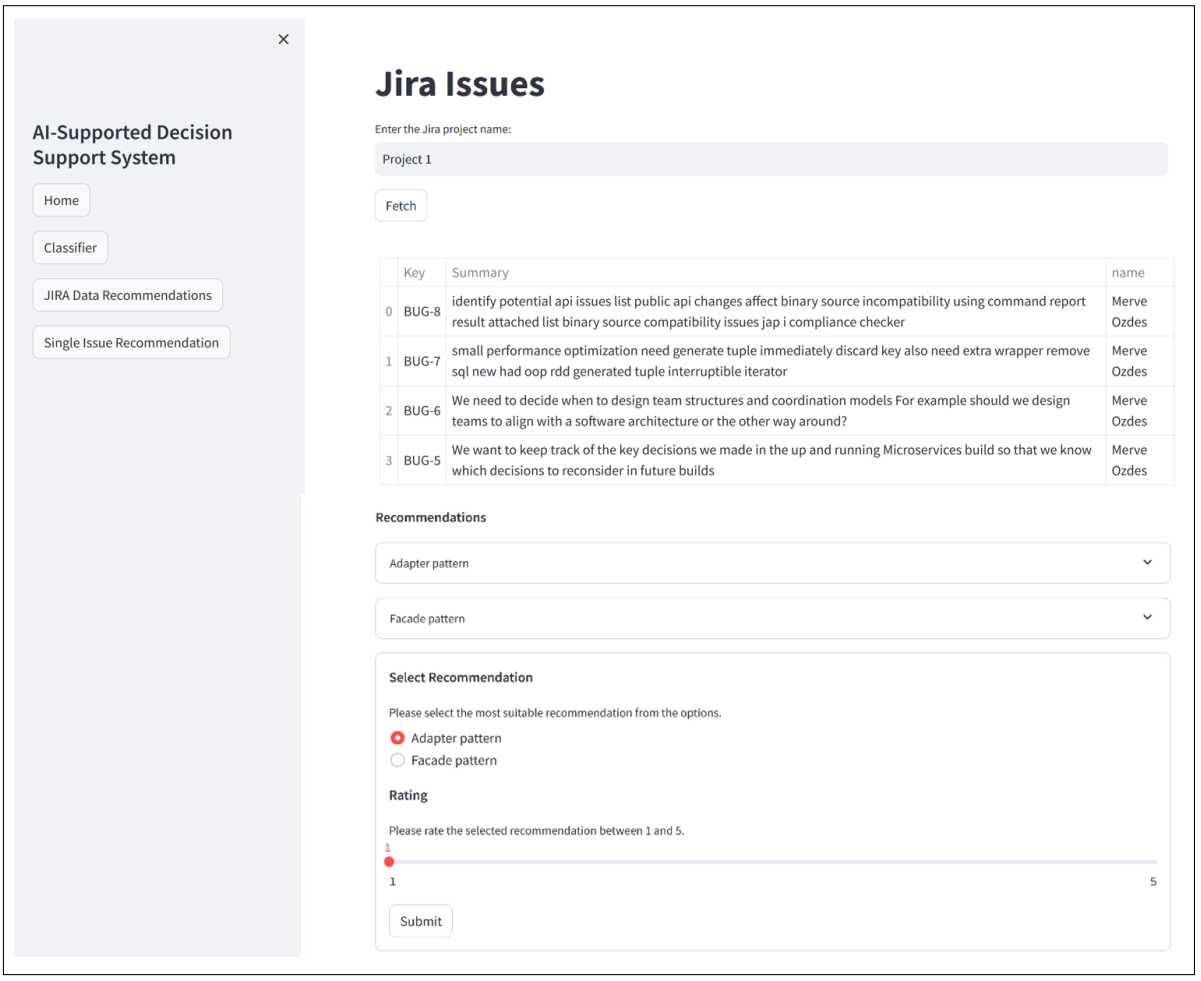


Figure 4.14 JIRA Data Recommendations page and system responses

The page shown in Figure 4.14 works with Jira issues. This page allows users to bring issues from a specific Jira project and examine potential design model recommendations based on the characteristics of each issue. Users can enter the name of the Jira project they are interested in (for example, ”Project 1”) and then click on the ”Fetch” button to fetch the relevant issues. Once fetched, the issues in the Jira project are displayed with their keys, summaries and other relevant details. Each issue is listed with a unique identifier (e.g. BUG-8, BUG-7) and a summary that provides a brief description of the issue. Only design related issues are considered and analyzed accordingly. Based on the analysis of the issue descriptions, the system suggests appropriate design models that could potentially solve or address the challenges highlighted in the issues. For example, the system may suggest

”Adapter pattern” or ”Facade pattern” depending on the nature of the problem described in the issue. Users can select the most appropriate design pattern from the suggested options. After selecting a pattern, users are asked to rate how appropriate they find the suggestion on a scale from 1 to 5, where 1 means least appropriate and 5 means most appropriate. After making a selection and providing a rating, users can submit their feedback by clicking on the ”Submit” button. This action finalizes their input and can contribute to the system’s learning and future recommendation accuracy.

Translated with DeepL.com (free version)

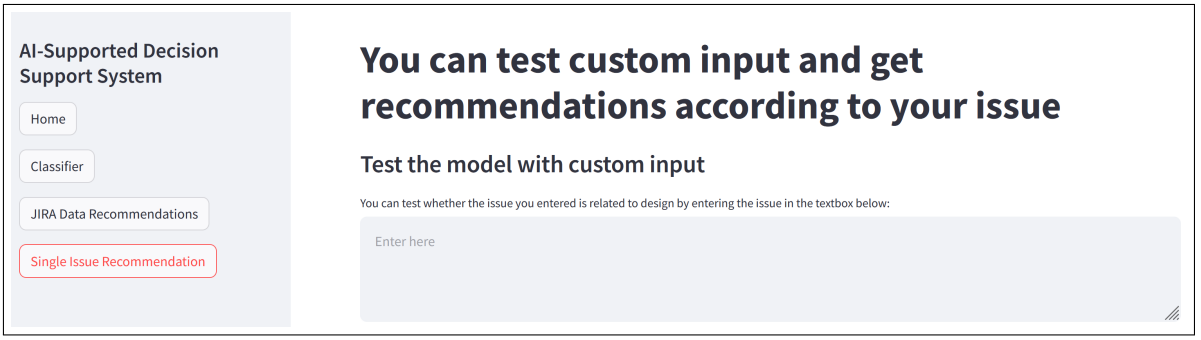


Figure 4.15 Single Issue Recommendations page of user interface

The single issue page displayed in Figure 4.16 allows users to test the model with custom input. When a user inputs an issue, the system first determines whether the issue is related to design. If the issue is design-related, the page displays relevant results using two local explanation methods: LIME and SHAP. These methods help illustrate why the model made its prediction about the issue being design-related. Additionally, the system recommends specific design patterns suitable for the entered issue. For the example given in Figure 4.16, ”State pattern” and ”Strategy pattern” are recommended. Users can then select from the recommended design patterns and provide a rating for the chosen pattern. The rating scale is from 1 to 5, allowing users to evaluate how well the recommended pattern suits their specific needs. This feedback mechanism helps refine the recommendations and enhance the system’s accuracy and relevance for future queries.

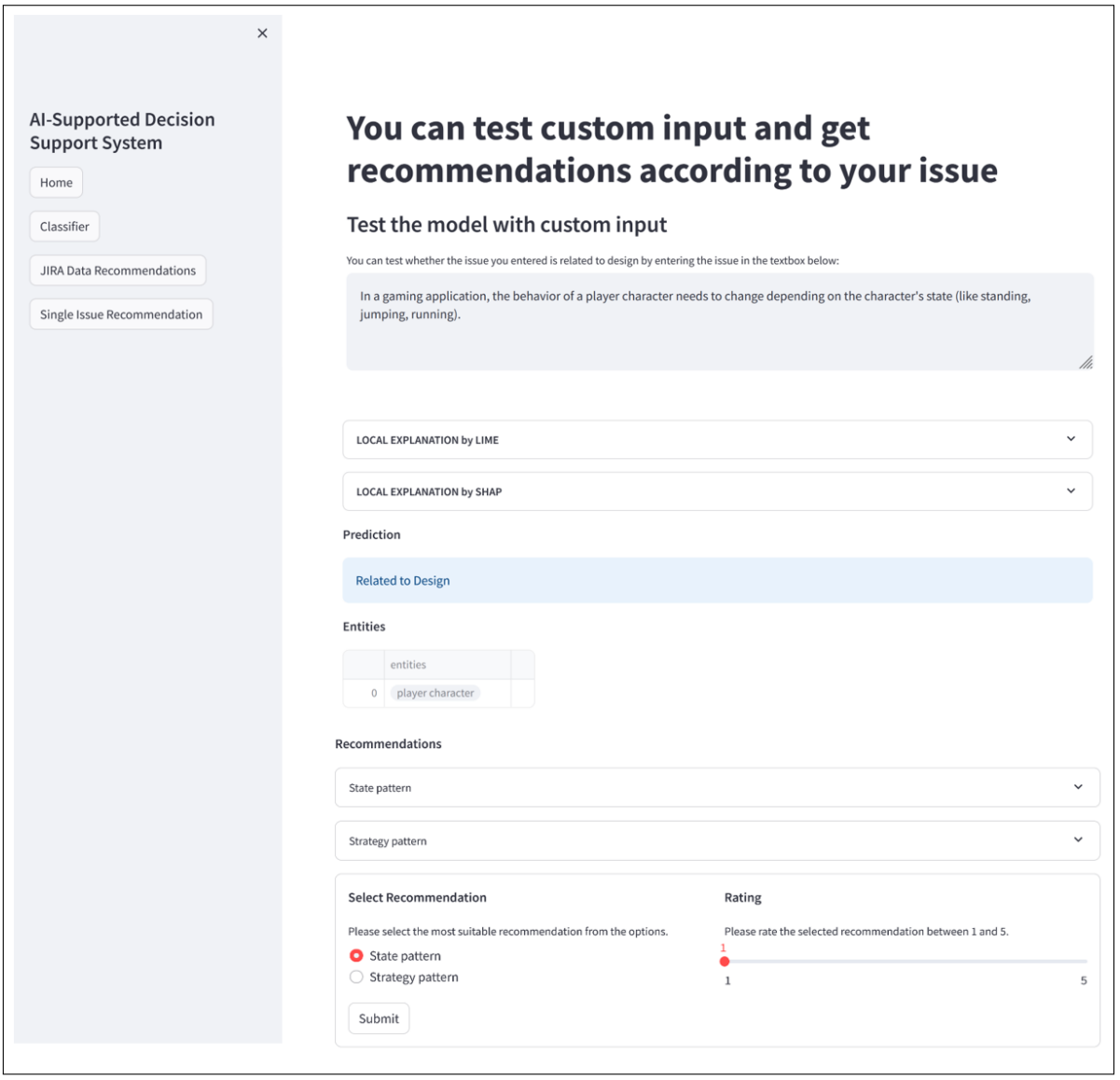


Figure 4.16 Single Issue Recommendations page and system responses

4.4. VALIDATION OF SOFAR-DSS

The validation process for SOFAR-DSS incorporated three main approaches: validation with Stack Overflow data, validation with design pattern cases from book and validation with experts. To understand the intricacies of the inputs, selected samples from the initial two validation techniques and a case instance for expert validation and their responses are presented.

4.4.1. Validation with Stack Overflow Data

Initially, we leveraged a validation approach centered around issues shared and discussed on the Stack Overflow and their corresponding solutions posted on Stack Overflow. We collected software development-related queries and their accepted solutions from this platform to test SOFAR-DSS. We specifically targeted questions that had received community endorsement through accepted answers. A preliminary filter of tags associated with 'software-design' and 'design-patterns' produced 279 questions. We meticulously examined these to select queries with comprehensive problem descriptions and community-approved answers. After a thorough selection process, we pinpointed 20 questions that closely matched the operational scope of SOFAR-DSS. These were then input into our system to evaluate the alignment of our system's suggestions with the accepted responses on Stack Overflow. The results were promising: SOFAR-DSS's recommendations corresponded with the accepted answers in 17 out of the 20 instances. This 85% confidence rate not only demonstrated SOFAR-DSS's proficiency in offering viable solutions for practical software design queries but also validated that its recommendations were consistent with community-vetted solutions, thus boosting the system's dependability and applicability. The validated questions and a comparison of Stack Overflow's accepted answers with SOFAR-DSS's recommendations are summarized in Table 4.4, presenting select examples for illustration. All the questions used for validation from Stack Overflow, along with their respective answers and the recommendations provided by SOFAR-DSS, are presented in Table 6.1 in Appendix A.

4.4.2. Validation with Design Pattern Cases from Book

As a subsequent step, a set of example cases related to software design patterns, derived from Shvets' book on design patterns [131], were manually entered into SOFAR-DSS. These cases presented typical challenges encountered in software architecture, along with a spectrum of applicable software design patterns. We meticulously inputted examples for each of the 22 design patterns into the system, except for the chain of responsibility pattern

Table 4.4 Sample Questions from Stack Overflow and their Accepted Answers Compared with SOFAR-DSS's Responses

Stack Overflow Question	**Accepted Answer on Stack Overflow for corresponding question**	**SOFAR-DSS's Recommendations**
I have some pre initialized objects of some class. These objects are heavy weight objects and each correspond to some configuration options specified by user. There will be exactly one instance corresponding to one configuration and same will be used every time.	Flyweight Design Pattern	Flyweight Design Pattern, Object Pool Design Pattern
I have developed a VSTO 4.0 add-in, designed for the integration of our business application into Microsoft Word. Code written in the making add-in, is a mini framework that I would like to re-use for a new add-in for Microsoft PowerPoint. The main problem is that the Word and PowerPoint interfaces (for example, Microsoft.Office.Interop.Word.Table and Microsoft.Office.Interop.PowerPoint.Table) do not have a common ancestor, but I need to create an general API for inserting, updating, tables and graphs, etc., which will be standardized to work with Word and PowerPoint objects	Adapter Design Pattern	Adapter Design Pattern, Strategy Pattern
An example could be pooling videos from different video-streaming services into a generalized model. In other words, each video-streaming service will have their own representation of a Video object with a different set of properties. So you want to gather these different constructs and aggregate them into a generalized Video object.	Composite Design Pattern	Builder Design Pattern, Adapter Design Pattern

due to its inadequate problem description, which led us to deactivate it. The examples were carefully chosen to cover the three main pattern categories—creational, structural, and behavioral—as defined by Shvets. By processing these instances, we could critically evaluate SOFAR-DSS's capability to offer solutions across varied architectural dilemmas. We then measured the system's output against the established patterns and best practices from the literature. SOFAR-DSS's accuracy in suggesting appropriate patterns was confirmed in 17 out of 23 cases. In one instance, the system did not categorize the issue as 'design related' and hence did not offer a recommendation, and for another case, it made no suggestion at all. This demonstrated the system's adeptness in adhering to recognized design pattern conventions and its effectiveness in tackling a broad array of architectural situations. The cases utilized for system validation have been disclosed. Table 4.5 lists some instances from Shvets's guide, alongside the corresponding recommendations from SOFAR-DSS. In Appendix B, Table 6.2 lists all instances from Shvets's guide, alongside the corresponding recommendations from SOFAR-DSS.

Table 4.5 Example Cases with Design Pattern and the SOFAR-DSS's Response

Example case from design pattern book	**Design pattern as outlined in the book for the corresponding example case**	**Recommended Design Patterns by SOFAR-DSS**
Application needs to manipulate a hierarchical collection of "primitive" and "composite" objects. Processing of a primitive object is handled one way, and processing of a composite object is handled differently. Having to query the "type" of each object before attempting to process it is not desirable.	Composite Design Pattern	Composite Design Pattern, Visitor Design Pattern
Application needs one, and only one, instance of an object. Additionally, lazy initialization and global access are necessary.	Singleton Design Pattern	Singleton Design Pattern, Factory Method Design Pattern
"Hardening of the software arteries" has occurred by using subclassing of an abstract base class to provide alternative implementations. This locks in compile-time binding between interface and implementation. The abstraction and implementation cannot be independently extended or composed.	Bridge Design Pattern	Strategy Design Pattern, Abstract Factory Design Pattern

4.4.3. Expert Validation and Feedback

To evaluate SOFAR-DSS thoroughly, we consulted with subject matter experts for their professional feedback.This process involved five specialists: four seasoned software architects with a decade of experience each, and a senior developer with nine years in the field. Before they began their assessment, we provided these experts with a clear understanding of the data intricacies by showcasing a range of pre-tested case studies sourced from both Stack Overflow discussions and established literature. Given the sensitivity of proprietary corporate data, the specific details of company-related inputs remain undisclosed in our documentation. These professionals also engaged with the system using sample inquiries from Stack Overflow, thereby gauging the system's applicability to actual industry challenges. Table 4.6 in our document illustrates an instance where an expert interacted with SOFAR-DSS, highlighting the system's response mechanism. This case study serves as a testament to the system's operational effectiveness when confronted with real-life problem statements. The system adeptly executes multiple functions: from classification and entity recognition to query response and pattern suggestion, offering users a cohesive and interactive experience that culminates in customized solution recommendations.

Table 4.6 SOFAR-DSS's responses according to the issue entered by user

Issue Entered	A broadcasted table will be used multiple times in a query. We should cache them avoid duplicated broadcasts.
Classification Result	Related to design
Entities	cache
Recommendations given by SOFAR-DSS	Observer pattern, Flyweight pattern

When an issue entered by a user is classified as 'design related' by the system, the user interface showcases the influence of individual words on this classification through LIME and SHAP visualizations.

SHAP (SHapley Additive exPlanations): SHAP is a game theory-based model interpretability tool that assigns each feature an importance value for a particular prediction [123]. Its approach is grounded in the concept of Shapley values from cooperative

game theory, which attributes a fair distribution of payoffs to players depending on their contribution to the total payout. In the context of machine learning, SHAP values provide insights into how much each feature contributes, positively or negatively, to the target variable. This is crucial for complex models, such as ensemble models or neural networks, where the relationship between input features and the prediction is not straightforward. SHAP not only helps in understanding individual predictions but also in analyzing the model's overall behavior. The visualizations generated by SHAP summarize the effects of features across multiple instances, making it a powerful tool for both local and global model explanation.

LIME (Local Interpretable Model-agnostic Explanations): LIME is a technique designed to explain individual model predictions in a way that humans can understand [122]. It works by creating a simple, interpretable model, such as a linear model, which approximates the predictions of the complex model locally. To do this, LIME perturbs the input data, generating a new dataset consisting of the perturbed samples and the corresponding predictions from the complex model. Then, it weights these new samples according to their proximity to the instance being explained and learns a simple model on this weighted dataset. The resulting model is interpretable as it usually relies on a smaller, more understandable set of features. The key idea is that even though the overall model may be complex and nonlinear, around a local point, it can be approximated well by something simpler. LIME can be applied to any model, making it model-agnostic, and it provides a way to understand why a model made a certain prediction, thereby increasing trust in the model's decision-making process.

LIME and SHAP visualizations are depicted in Figures 4.17 and 4.18 for the case described in Table 4.6. Detailed commentary for each figure follows below. The LIME visualization (Figure 4.17) uses red and green to signify which elements (words, in this context) have the most sway in the model's decision-making process. The graph reveals that the length of the bars corresponds to the extent of influence, with green representing positive and red indicating negative impacts. For instance, the word 'use' is shown to have a significant positive effect on the model's prediction, as evidenced by a long green bar. Conversely,

'avoid' also contributes positively, though to a lesser degree than 'use', while 'table' appears to exert a negative influence. Other terms are seen to have negligible or neutral effects. The SHAP summary plot in Figure 4.18 offers a graphical representation of each feature's effect on the model's output for a particular prediction class, labeled 'Design Related'. Here, individual points signify the SHAP value for a feature in a single instance, reflecting its contribution to the model's prediction. These values' placement on the X-axis denotes the strength and direction of their impact. Features leading to a higher prediction appear with positive SHAP values, and those leading to a lower prediction show negative values. 'Use' stands out with a pronounced positive impact, nudging the model towards a class 1 (related to design) prediction. In contrast, 'multiple' has a lesser negative effect, indicated by a red bar, moving the prediction slightly away from class 1. Blue dots for words like 'table' and 'time' denote a neutral or variable influence on the model's outcome. Features positioned to the right side of the zero line contribute to a class 1 classification, while those on the left side detract from it. When examining both graphs in Figure 4.17 and Figure 4.18, they seem to provide congruent insights, although the specific words with negative influence differ—'table' in the LIME graph versus 'multiple' in the SHAP graph. Using the LIME approach necessitates a critical approach to its results due to inherent constraints. LIME requires the user to specify the number of top features to focus on, which is inherently subjective [132]. Moreover, it creates neighboring text data by randomly omitting words, possibly leading to illogical strings of words that poorly represent actual data scenarios. This randomness does not reflect the true distribution of feature values or the clustering of class labels present in the data [133]. Hence, the model's behavior is approximated based on these distorted values, which might result in explanations that don't accurately portray real situations. While LIME can provide useful insights, its important to interpret its results cautiously, keeping in mind the possibility of skewed interpretations because of these methodical limitations.

The interface contains an area where users can rate the recommended recommendations following classification, entity extraction and recommendation sections. From the suggested options, users select and rate the design pattern and then submit their comments. In depth

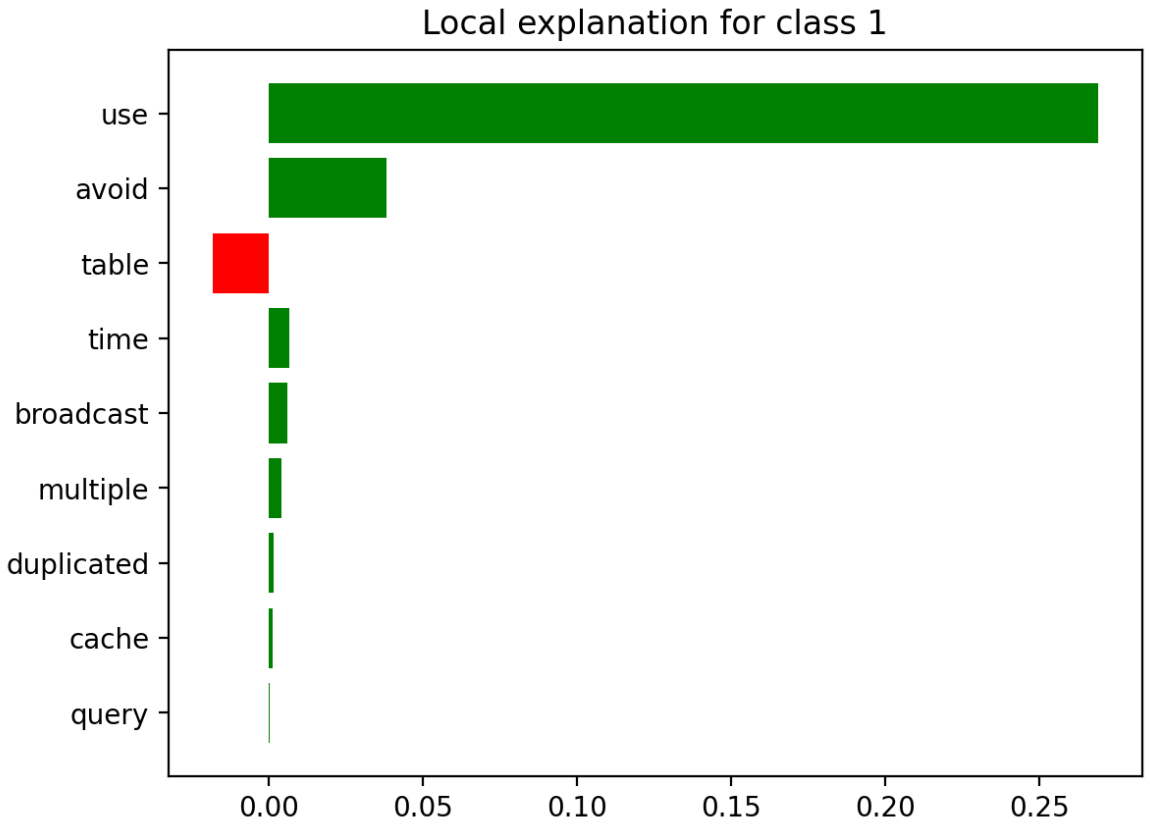


Figure 4.17 Local explanation of text classification with LIME on individual instance for class 1 (related to design)

analysis of the system's performance and usability has been carried out by experts through collective feedback. We have conducted a survey of 11 questions to collect this feedback on a systematic basis. A Likert scale was used in six of these questions to assess user satisfaction with various aspects, such as ease of use, interface design, accuracy, reliability of recommendations, and the impact of the system on decision making. Table 6 gives an overview of the responses to Likert's questions. Further questions were opened to collect input on system improvements, potential new features and any problems that might arise in the course of use. Three of the experts indicated that they intend to apply SOFARDSS in their own companies. The replies indicated that the system was easy to use for all experts, and that they would recommend it to others. The recommendations were considered to be very accurate and adequate. The system has been considered to be effective as regards decision making, although some feedback was received on improving it. The addition of

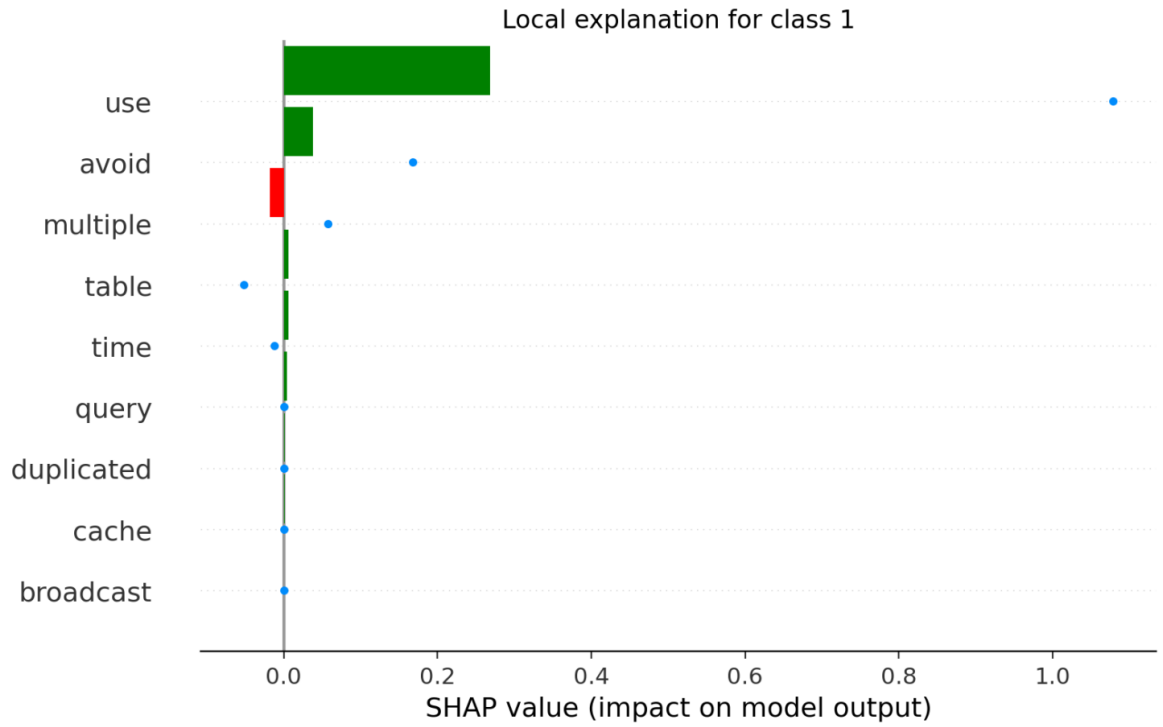


Figure 4.18 Local explanation of text classification with SHAP on individual instance for class 1 (related to design)

implementation details, UML diagrams and code for recommendations, explanations on why they have been recommended, as well as the possibility of integrating this system into project management tools like Jira were all suggestions for improvement. The requirement of detailed task inputs to achieve optimum results has been identified as a major challenge.

Consequently, considerable insight was gained through a thorough validation of SOFAR-DSS. Its effectiveness and reliability were confirmed by the alignment of the system's recommendations with recognized solutions for Stack Overflow, as well as its design patterns. In addition, the practical applicability of this system has been reinforced by favorable comments from industry experts that highlighted its ease of use, accuracy, and usefulness for decision-making.

The suggestions for improvement have also provided valuable directions for future enhancements, motivating SOFAR-DSS will remain relevant and effective in evolving software architecture scenarios. Overall, this validation process has robustly provided

Table 4.7 Likert questions for evaluation of system and the Likert scores of the answers for all experts (E1-E5)

Question	E1	E2	E3	E4	E5	Mean	Median	Mode
How easy was the system to use? (1-Very Difficult, 5-Very Easy)	5	5	5	5	5	5	5	5
Was the interface design and navigation intuitive? (1-Strongly Disagree, 5- Strongly Agree)	5	5	4	5	5	4.8	5	5
How accurate and appropriate did you find the recommendations provided by the system? (1-Not Accurate at All, 5-Very Accurate)	4	4	4	5	4	4.2	4	4
How reliable did you find the system's recommendations? (1- Not Reliable at All, 5-Very Reliable)	4	5	4	4	4	4.2	4	4
How useful was the information provided by the system in your decision-making process? (1- Not Useful at All, 5-Very Useful)	3	4	3	4	4	3.6	4	4
How often do you plan to use the system? (1-Never, 5-Very Frequently)	3	3	2	4	3	3.0	3	3
Would you recommend the system to your colleagues? (Yes-Y, No-N, Maybe-M)	Y	Y	M	Y	Y			

evidence that SOFAR-DSS is a credible candidate tool for software architects and developers. The upgrade recommendations have also yielded important avenues for future development, ensuring that SOFAR-DSS will continue to be applicable and efficient in changing software architecture circumstances. All things considered, SOFAR-DSS is a reliable candidate tool for software architects and engineers, as demonstrated by the solid proof this validation procedure has produced.

4.5. Threats to Validity

Several potential threats to validity have been found and considered during this study on the use of a QA Model and DBpedia ontology, which recommended software architecture patterns. These threats and the steps taken to mitigate them are dealt with in this section.

Internal Validity: The recommendations made by the QA model can be affected by biases in the training data. Efforts have been made to diversify the training set, but some residual biases may remain. The classification of problems and subsequent entity extraction depends on the quality of the input data. Any inaccuracies at this stage can cascade through

the system. To improve data accuracy, we performed rigorous preprocessing and applied numerous iterations of manual validation. The selection of trust and support parameters for DBpedia Spotlight was based on preliminary experiments designed to balance precision and recall. These parameters may need to be further refined for different datasets or domains. There was a potential threat of bias in the selection of questions and answers from Stack Overflow. This could mean that our validation process may only represent certain types of problems or user perspectives. To mitigate this, we used a random selection process to cover a range of questions and provide a more representative sample of real-world software development scenarios.

External Validity: Our presented results may not represent all architectural design scenarios. We have limited this potential threat by including a wide range of topics and models from different sources. The effectiveness of the methodology in different areas of software engineering has not been extensively tested. Future work could broaden the scope to include more diverse domains.

Construct Validity: There may be inconsistencies between the conceptual definitions of the entities in our study and their real-world counterparts. To mitigate this, we closely aligned our definitions with established software engineering terminology. The choice of DBpedia Spotlight for entity extraction was based on its extensive use in this domain, even though it is not fully aligned with software architecture-specific jargon. A thorough review of the literature supported its application in this context.

4.6. Evaluation and Discussion

Our methodology for combining the QA model with the DBpedia ontology underwent a thorough evaluation. We collected and classified issues from JIRA and applied entity extraction to improve the performance of the QA model. We fine-tuned the precision of entity recognition with DBpedia Spotlight to reach an ideal balance, thus guaranteeing both high reliability and broad coverage. We measured the effectiveness of our classification models using traditional metrics such as accuracy, precision, recall, and F1 score. These models

include both traditional machine learning models KNN, SVM, RF, XGBoost and LGBM, and deep learning models BERT and LSTM. The model selection process was data-driven and cross-validated to ensure robustness. The LSTM classifier was particularly adept at managing the complexities of software architecture problems. We used SHAP and LIME, both explainable artificial intelligence (XAI) techniques, to elucidate the operation of our classification algorithms. The integration of XAI methods allowed us to provide users with transparent and understandable explanations for classification decisions. These techniques help to understand the rationale for classification by identifying specific words or phrases in the text that most influenced the model's decision. The crucial entity extraction step of our methodology was supported by DBpedia Spotlight's deep semantic analysis capability. The extracted entities played a crucial role in the creation of structured queries that fed the QA model, establishing a direct link between the issue reports and the corresponding architectural models. This harmonious interaction between entity extraction and the QA model underpins the effectiveness of our approach. The practical results of our system have been significant, demonstrating its capacity to markedly improve decision making in software architecture and facilitate more streamlined and enlightened pattern selection. The adaptive framework of the system points to its potential to evolve with the field by continuously incorporating new patterns and architectural insights.

While the SOFAR-DSS introduced here serves as a navigation aid for decisions in software architecture, it is imperative that users accept the constraints of the system and exercise discretion under certain conditions. The decision-making capability of the system depends on the detail provided in the user's input; therefore, vague or ambiguous requirements may result in less accurate recommendations. To achieve the most precise and useful results, users are encouraged to articulate their architectural issues in clear and relevant key phrases. Furthermore, the effectiveness of the DSS depends on the extent to which it represents the domain of the underlying database and algorithms, which may not cover the full range of emerging technologies and methodologies. It is crucial that users supplement the DSS's recommendations with experienced human expertise, especially in complex cases that may require an assessment beyond what the system currently offers. Users should also be aware

that the DSS functions as an auxiliary tool and is not intended to replace the complex and critical decision-making processes carried out by experienced architects.

5. SUMMARY

This thesis topic was chosen based on the observation of the ongoing challenges and inefficiencies in architectural decision making in software development. The complexity of this process, characterized by a large number of options, the need for rapid decision making and rigorous documentation requirements, poses significant challenges for software architects. Such challenges not only hinder the process, but also affect the quality and effectiveness of the software systems developed. Therefore, we aim to explore innovative solutions that can improve the architectural decision-making process. This led to the development of SOFAR-DSS, a tool designed to assist architects by simplifying decisions and increasing the precision and relevance of software design recommendations. The motivation stemmed from the desire to provide a robust framework that can alleviate the burdens faced by architects and improve the overall decision-making environment in software architecture.

In this context, this thesis started with a semi-structured exploratory study and an online questionnaire aiming to uncover the realities of architectural decision-making in the field, the influential and determining factors within this process, and the challenges that are encountered. We targeted active software professionals involved in architectural decisions, whether currently or in the past. The semi-structured exploratory study was conducted with 9 experts from Turkey involved in architectural decision-making, while the survey was conducted with 101 experts from around the world. The data was subject to both quantitative and qualitative analysis techniques.

Valuable insights were obtained about the actual practices of decision-making within organizations and by architects, the documentation and accessibility of these decisions, the factors that pose challenges and hold significance in decision-making, the composition and methods of the decision-making teams, and the aspects in need of enhancement within the decision-making process. The findings reinforce the critical role that the process of making software architecture decisions plays within the industry. This is evidenced by

participant feedback and their responses to open-ended questions within the survey. Our primary observations indicate that decisions are typically recorded and made collaboratively in teams. Brainstorming and consensus, which allow for open expression of ideas by all team members, are favored approaches. Reviewing and discussions are common practices for tracking interrelated decisions, defining architecturally significant requirements, and confirming decisions made.

These outcomes highlight the value of discussion and information exchange for reaching decisions in the architectural decision-making process. While some organizations follow a systematic approach for each phase of this process, others do not adhere to any formal methodology.

Diverging from previous survey research in software architecture, we statistically evaluated responses from different groups (for instance, whether decisions are documented, whether Agile methodologies are used) to the survey questions. Using the Mann-Whitney U test and correlation analyses, we assessed the existence of statistically significant differences in responses. The findings from these tests revealed significant variances between the groups concerning both technical and social challenges faced in making architectural decisions, as well as in responses to the factors affecting these decisions. Notably, peer pressure in projects directed by an individual differed significantly from that in team-driven projects, suggesting that an increase in the number of decision-makers can lead to more disagreements. However, when comparing Agile versus non-Agile projects, and projects where decisions are documented against those that are not, no significant differences were found in terms of the factors influencing architectural decisions. This implies that the development lifecycle model and documentation status do not sway the decision-making in architecture.

From our research findings, we deduce that effective communication holds paramount importance in the decision-making process, as also inferred from the responses to questions 18b and 28. This stakeholder communication can be facilitated by robust architectural documentation practices. To this end, the utilization and customization of documentation templates from literature could be advantageous and tailored to best suit and serve

the organizational needs. Our study not only underscores the necessity of effective communication but also the importance of a solid documentation process to enable it.

The survey predominantly included general questions reflecting the participants' past experiences. In future work, the survey could be tailored with more project-specific inquiries. Additionally, investigating the relationship between software architecture decisions and company or project characteristics could contribute significantly to the field of architectural decisions. A study examining the link between technically unsuccessful projects and their architectural decisions would also be enlightening, offering insights into the underlying causes of project failures and the influence of architectural decisions. A forthcoming initiative planned by the authors of this study is to develop a decision support system aimed at mitigating uncertainties and challenges in the architectural decision-making process, thereby streamlining it. This system, which is currently in development, seeks to enhance the success rates of decision-making in software architecture through more systematic documentation and improved traceability of decisions. The insights garnered from this study will be pivotal in the creation of this decision support system and its accompanying tools. Lastly, another potential avenue for future research is delving into the improvements in the decision-making process as indicated by the study respondents, necessitating research focused on understanding and addressing the problems and inefficiencies highlighted by this study.

The insights from these studies have significantly improved our understanding of the architectural decision-making process. Our findings show that this process is fraught with challenges, making decision making increasingly difficult, especially when decision makers are faced with insufficient information or a large number of alternatives.

To facilitate this process and ease the burden on architects, we introduced SOFAR-DSS. The motivation behind the design of SOFAR-DSS was influenced by the feedback received from people involved in architectural decision-making in various companies, especially regarding problems with documentation. It was often mentioned that documentation was

often inadequate or seen as an additional burden. Our research revealed that decisions were often made on the basis of meeting notes or personal records.

Based on this, we aimed to design SOFAR-DSS to assist decision makers by integrating issue tracking systems to retrieve relevant issues and propose design patterns accordingly. This approach aims to prevent decision makers from spending excessive time evaluating different alternatives and thus facilitate decision making in architectural studies. It is expected that this system will not only simplify the process but also improve the quality and efficiency of architectural decisions by providing timely and relevant information and recommendations.

As the final study of this thesis, we introduce SOFAR-DSS, a methodology designed to refine the process of recommending software design patterns. Our approach was driven by the need to make architectural pattern recommendations more pertinent to the distinct challenges encountered during software design and development. Recognizing the importance of customizing architectural advice to resolve specific issues faced by software architects or decision-makers, we utilized a blend of question-answering models and the extensive repository of knowledge contained within the DBpedia ontology.

Our methodology encompassed several critical stages, each contributing to a systematic and effective design pattern recommendation process. We initiated by gathering issues from the JIRA tracking system and employed sophisticated classification algorithms to ensure precise categorization of these issues, distinguishing between design-related and unrelated concerns. To provide transparent and comprehensible explanations for our models' classification decisions, we integrated explainability methods such as SHAP and LIME. These approaches shed light on the influence of particular features on classification results, thus improving the transparency and reliability of the recommendation process.

To enrich the issue data further, we harnessed the power of DBpedia Spotlight for robust entity recognition. This step was instrumental in extracting significant information and entities from the issue narratives. The integration of external knowledge from DBpedia supplemented our comprehension of the issues. Utilizing the extracted issues and entities, we deployed a question-answering model to deliver software pattern suggestions that were

customized to specific requirements. To corroborate these suggestions and enhance them further, we retrieved corresponding software patterns from DBpedia using SPARQL queries, capitalizing on the extracted entities.

By combining these innovative technologies and methods, our research has considerably improved the identification and recommendation of pertinent architectural patterns, marking a significant advancement in intelligent decision-support systems within software architecture. Our validation process was crucial for determining the effectiveness of SOFAR-DSS. First, by using verified solutions from Stack Overflow queries, we thoroughly tested our system against practical situations, ensuring the accuracy and practicality of our recommendations. Second, by incorporating various software pattern cases into our system, we further validated its capacity to recommend architectural solutions. Lastly, we gathered insights from five expert software architects who utilized SOFAR-DSS.

Our work underscores the potential of semantic technologies and sophisticated classification methods to tackle intricate issues across diverse domain-specific applications. This contribution not only assists software architects in making better-informed decisions but also demonstrates the versatility of semantic technologies in complex, domain-specific applications.

Future research directions in this area include broadening the dataset to improve the generalizability of our findings. Subsequent studies could benefit from including a wider array of software project types from diverse settings to enhance the dataset's richness. Moreover, the integration of SOFAR-DSS with additional models and ontologies from the field of software architecture could substantially elevate the quality and detail of the system's recommendations. Another promising avenue is to develop a mechanism for continuous learning, which would allow the system to evolve and refine its suggestions based on new information, emerging trends, and user feedback. Such an approach could make the model more resilient and adaptable over time, boosting its long-term utility in the ever-changing landscape of software development. Furthermore, an extension of our work could involve providing users with explanations for the recommended software design

patterns, thereby increasing the decision-making process's transparency and giving users a deeper understanding of each recommendation's rationale, ensuring the tool's outputs align closely with the unique demands and constraints of their projects.

6. CONCLUSION

6.1. Contributions

In this thesis, we have concentrated on identifying enhancements that can alleviate the challenges decision-makers encounter during the architectural decision-making process. As a foundational step, we conducted a comprehensive survey to identify factors and challenges prevalent in architectural decision-making. It targets a diverse spectrum of industry professionals and academics. Our reach was both broad and deep: we distributed over a thousand emails, as well as LinkedIn and Facebook messages directly to individuals on the ground. Our efforts went beyond these direct communications, leveraging the power of the digital network through strategic posts on forums and social media platforms as well as announcements in private groups to maximize visibility and engagement. We also leveraged network influence by encouraging practitioners to disseminate the survey to their professional circles, thereby increasing our reach through their personal endorsement. This multi-pronged outreach strategy resulted in a significant pool of 101 private individuals completing the survey, providing us with a wealth of insight. This survey distinguishes itself from prior studies by reaching a broader participants and analyzing the factors and challenges across all stages of the architectural decision-making process, from the identification of project requirements to the final steps. Unlike other studies that may focus on a single phase, our approach involved examining each process in detail, posing questions to participants that would yield insights into each specific stage.

In particular, our analysis shows that knowledge and insights gained from architectural decisions in one project can be instrumental in informing and improving decisions in subsequent projects within the same company. This cross-project application of architectural knowledge represents an important area for future research by suggesting a model where learnings from one context can directly benefit another, similar to knowledge transfer practices observed in other areas of software engineering. By systematically documenting decisions and their outcomes, companies can create a store of knowledge that can serve

as a valuable reference for future projects. This helps not only to avoid past mistakes but also to repeat success, thus streamlining the decision-making process and improving project outcomes. Moreover, such accumulated cross-project knowledge may serve as a basis for any kind of automated decision support systems

Our findings illuminate the complex interplay between decision-making practices and the broader objectives of software architecture, underscoring the critical role of documentation and collaborative decision-making. Notably, the preference for team-based decisions and the reliance on brainstorming and consensus underscore the value of diverse perspectives and collective intelligence in navigating the intricacies of software architecture. This collaborative ethos not only enriches the decision-making process but also fosters a culture of shared responsibility and mutual understanding among team members.

The findings from the survey underscored the necessity of a tool that supports decision-makers throughout the software architecture decision-making process. To address this need, we developed an intelligent decision support system that utilizes both question-answering capabilities and ontology to provide specific and accurate recommendations to decision-makers. Our work uniquely combines different methodologies, leveraging the strengths of each approach to create a comprehensive system. Additionally, we have integrated explainable artificial intelligence (XAI) into our system to enhance transparency and reliability from the end-user's perspective. This inclusion allows users to understand how the model classifies issues and the influential words that drive these classifications, fostering a more transparent system.

We validated our work through three distinct methods, demonstrating its applicability and accuracy in providing recommendations within the real world. This multi-faceted validation approach has shown that our system has the potential to succeed across various domains, indicating its robustness and versatility.

6.2. Challenges

During the development of this thesis, several challenges emerged that required careful consideration and strategic problem solving. These challenges not only tested the robustness of our methodologies, but also provided valuable learning experiences that enriched the overall research.

- Participation and Response Rates: One of the main challenges was to ensure a high level of participation for our survey. Despite an extensive outreach strategy that included direct and indirect communication via email, LinkedIn, Facebook and various online platforms, response rates posed a significant barrier. While our multi-pronged approach ensured that 101 full responses were received, this number represents only a fraction of the total reach and suggests the need for more different engagement strategies to increase participation in future studies.

- Diversity of Perspectives: The diversity of survey responses was another challenge. Given the complexity and broad scope of the architectural decision-making process, it was critical to gather insights from a wide range of professionals from different areas of expertise, seniority levels, and geographic locations. This diversity was necessary to accurately project the results onto a specific subset of professionals or a specific set of architectural practices.

- Complexity of Data Analysis: The survey revealed a rich dataset that required rigorous analysis. The complexity and granularity of the data created a challenge in identifying the most relevant factors and challenges influencing the architectural decision-making process. We needed to develop a robust analytical framework that could transform the vast amount of information into actionable insights.

- Development of the Decision Support System: The creation of the intelligent decision support system posed numerous technical challenges, from integrating question answering models with the DBpedia ontology to incorporating XAI for transparency.

Balancing accuracy, usability and explainability required a nuanced approach to system design and continuous iteration.

- Finding the Data Set: Finding the data that will form the knowledge base of the decision support system was one of the biggest challenges we faced in this thesis. Both the confidentiality policies of the companies and the lack of systematic documentation in the architectural decision-making process led to this challenge. For this reason, we utilized the dataset of a study in the literature to create the knowledge base in our system.

- Validation of Recommendations: Validating the recommendations provided by the system required a rigorous and multifaceted approach. We needed to demonstrate the effectiveness of the system in different scenarios and domains, which required developing a validation process that could be universally applied and at the same time take into account the specific nuances of each case.

- Future-proofing the System: The final challenge is to ensure that going forward, as the software architecture evolves, the system remains current and useful. Continuous learning mechanisms need to be developed so that the system can adapt to new models, technologies and user feedback and remain relevant over time.

6.3. Constraints

The works conducted in this thesis, encompassing both the survey study and the development of the SOFAR-DSS, faced various constraints that significantly influenced the scope and depth of our research. First, despite extensive outreach efforts through email, social media, and professional networks, the response rate for our survey was lower than anticipated. This limitation in response rates severely restricted the scope of data available for analysis, thus significantly affecting the data collection phase and reducing the generalizability of our results across different demographic and professional groups. The limited data not only restricted the statistical power of our analyses but also reduced the diversity of insights we could draw regarding the architectural decision-making process across various industries.

Moreover, effectively using the SOFAR-DSS requires precise and comprehensive articulation of the issues at hand. This necessitates incorporating additional expressions and details to accurately frame the problem, adding another layer of complexity to the use of the system.

The rich dataset required a sophisticated analytical framework to properly extract actionable insights, necessitating extensive resources in terms of time and expertise.

Additionally, we encountered constraints in sourcing the data that would form the knowledge base of the decision support system. Due to companies' confidentiality policies, we were unable to access corporate data directly, which posed substantial challenges in data acquisition. These difficulties were compounded by difficulties in obtaining a robust data set due to privacy policies and the general lack of systematic documentation in this area. Many organizations do not maintain comprehensive records of their decision-making processes, which limited our ability to validate and enrich our findings with real-world data.

These constraints did not merely limit our research; they also illuminated critical areas for future improvement and innovation in the field. For instance, the challenges associated with data access and analysis highlighted the need for more collaborative relationships between academia and industry. Such partnerships could facilitate better data sharing agreements and improve the documentation practices within companies. Furthermore, the insights gained from navigating these constraints have underscored the importance of developing more robust methodologies in future studies that can accommodate and leverage incomplete or limited data sets more effectively.

6.4. Further Analysis

The development of the SOFAR-DSS represents a significant step forward in augmenting the cognitive capabilities of decision-makers in the architectural decision-making process. By leveraging a question-answering model and the DBpedia ontology, the system has demonstrated its potential to streamline complex cognitive tasks. The comprehensive

survey conducted prior to the system's development has provided valuable insights into the challenges and influential factors within the architectural decision-making process.

Going forward, our future work will focus on several key areas to enhance the SOFAR-DSS further. Firstly, we aim to expand the knowledge base of the system by integrating more diverse data sources, which will enrich the ontology and provide more comprehensive support for a wider range of architectural decisions. Secondly, we plan to implement machine learning algorithms that can learn from past decisions to offer increasingly relevant and personalized recommendations.

Another avenue for future work is the refinement of the natural language processing capabilities of the SOFAR-DSS. By incorporating advanced semantic analysis and understanding, the system can become more adept at interpreting the nuances of user queries, leading to more accurate and contextually appropriate responses.

Additionally, we recognize the importance of user experience in the adoption of decision support systems. Therefore, we will be conducting user-centered design studies to improve the system's interface and interaction mechanisms, making it more intuitive and user-friendly.

Furthermore, we intend to enhance the system by incorporating design styles—subcategories within the design patterns we currently recommend. By reflecting the differences between design patterns and design styles, this expansion will not only enhance the system's capability in handling decisions at various levels but also support higher-level strategic decisions. We also aim to extend our system's capabilities to support higher-level strategic decisions, making it a more comprehensive tool for project planning and architecture design.

Finally, to address the scalability and evolving nature of software architecture, we will work on developing adaptive algorithms that can incorporate the latest industry trends and feedback from the community of practitioners. Through continuous learning and adaptation, the SOFAR-DSS will aim to remain at the forefront of supporting architectural decision-making in an ever-changing technological landscape.

In sum, the future work for SOFAR-DSS is geared towards making it more robust, intelligent, and user-centric, thereby ensuring that it remains an essential tool for architects and decision-makers in the field of software architecture.

REFERENCES

[1] Nick Rozanski and Eoin Woods. *Software systems architecture: working with stakeholders using viewpoints and perspectives*. Addison-Wesley, **2012**.

[2] Anton Jansen and Jan Bosch. Software architecture as a set of architectural design decisions. In *5th Working IEEE/IFIP Conference on Software Architecture (WICSA'05)*, pages 109–120. IEEE, **2005**.

[3] Len Bass, Paul Clements, and Rick Kazman. *Software architecture in practice*. Addison-Wesley Professional, **2003**.

[4] Dan Tofan, Matthias Galster, and Paris Avgeriou. Difficulty of architectural decisions–a survey with professional architects. In *Software Architecture: 7th European Conference, ECSA 2013, Montpellier, France, July 1-5, 2013. Proceedings 7*, pages 192–199. Springer, **2013**.

[5] Hans Van Vliet and Antony Tang. Decision making in software architecture. *Journal of Systems and Software*, 117:638–644, **2016**.

[6] Olaf Zimmermann. Architectural decisions as reusable design assets. *IEEE software*, 28(1):64–69, **2010**.

[7] Paul Clements, Rick Kazman, Mark Klein, Divya Devesh, Shivani Reddy, and Prageti Verma. The duties, skills, and knowledge of software architects. In *2007 Working IEEE/IFIP Conference on Software Architecture (WICSA'07)*, pages 20–20. IEEE, **2007**.

[8] Matthew R McBride. The software architect. *Communications of the ACM*, 50(5):75–81, **2007**.

[9] Rick Kazman, Mark Klein, Mario Barbacci, Tom Longstaff, Howard Lipson, and Jeromy Carriere. The architecture tradeoff analysis method. In *Proceedings. fourth ieee international conference on engineering of complex computer systems (cat. no. 98ex193)*, pages 68–78. IEEE, **1998**.

[10] Rick Kazman, Jai Asundi, and Mark Klein. Quantifying the costs and benefits of architectural decisions. In *Proceedings of the 23rd International Conference on Software Engineering. ICSE 2001*, pages 297–306. IEEE, **2001**.

[11] Mikael Svahnberg, Claes Wohlin, Lars Lundberg, and Michael Mattsson. A quality-driven decision-support method for identifying software architecture candidates. *International Journal of Software Engineering and Knowledge Engineering*, 13(05):547–573, **2003**.

[12] Francisco Montero and Elena Navarro. Atrium: Software architecture driven by requirements. In *2009 14th IEEE International Conference on Engineering of Complex Computer Systems*, pages 230–239. IEEE, **2009**.

[13] Saheed Abiola Busari. *Modelling and analysing software requirements and architecture decisions under uncertainty*. Ph.D. thesis, UCL (University College London), **2019**.

[14] Marco Barenkamp, Jonas Rebstadt, and Oliver Thomas. Applications of ai in classical software engineering. *AI Perspectives*, 2(1):1, **2020**.

[15] Lei Wang. Ai in software engineering: Case studies and prospects. *arXiv preprint arXiv:2309.15768*, **2023**.

[16] Fahad H Alshammari et al. Trends in intelligent and ai-based software engineering processes: A deep learning-based software process model recommendation method. *Computational Intelligence and Neuroscience*, 2022, **2022**.

[17] Ipek Ozkaya. The next frontier in software development: Ai-augmented software development processes. *IEEE Software*, 40(4):4–9, **2023**.

[18] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick Van Kleef, Sören Auer, et al. Dbpedia–a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic web*, 6(2):167–195, **2015**.

[19] Dbpedia ontology. `https://www.dbpedia.org/resources/ontology/`.

[20] David Gunning and David Aha. Darpa's explainable artificial intelligence (xai) program. *AI magazine*, 40(2):44–58, **2019**.

[21] Merve Ozdes, Oumout Chouseinoglu, and Ayca Kolukısa Tarhan. Yazılım projelerinde mimari karar alma sürecini etkileyen faktörler ve karşılaşılan zorluklar: Keşifsel bir Çalışma. In *2020 8. Ulusal Yazılım Mühendisliği Konferansı*. TOBB ETU, **2020**.

[22] Merve Ozdes Demir, Oumout Chouseinoglu, and Ayca Kolukısa Tarhan. Sofar-dss: An advanced decision support system for architectural design patterns using openai and dbpedia. *Expert System with Applications*, **2024**.

[23] Davide Falessi, Muhammad Ali Babar, Giovanni Cantone, and Philippe Kruchten. Applying empirical software engineering to software architecture: challenges and lessons learned. *Empirical Software Engineering*, 15:250–276, **2010**.

[24] Mary Shaw and David Garlan. *Software architecture: perspectives on an emerging discipline*. Prentice-Hall, Inc., **1996**.

[25] Philippe B Kruchten. The 4+ 1 view model of architecture. *IEEE software*, 12(6):42–50, **1995**.

[26] Paul Clements, David Garlan, Reed Little, Robert Nord, and Judith Stafford. Documenting software architectures: views and beyond. In *25th International Conference on Software Engineering, 2003. Proceedings.*, pages 740–741. IEEE, **2003**.

[27] Neal Ford and Mark Richards. *Fundamentals of software architecture*. O'Reilly Media, Incorporated, **2020**.

[28] Manoj Bhat, Klym Shumaiev, Andreas Biesdorf, Uwe Hohenstein, and Florian Matthes. Automatic extraction of design decisions from issue management systems: a machine learning based approach. In *Software Architecture: 11th European Conference, ECSA 2017, Canterbury, UK, September 11-15, 2017, Proceedings 11*, pages 138–154. Springer, **2017**.

[29] Rafael Capilla, Anton Jansen, Antony Tang, Paris Avgeriou, and Muhammad Ali Babar. 10 years of software architecture knowledge management: Practice and future. *Journal of Systems and Software*, 116:191–205, **2016**.

[30] Anton Jansen, Jan Van Der Ven, Paris Avgeriou, and Dieter K Hammer. Tool support for architectural decisions. In *2007 Working IEEE/IFIP Conference on Software Architecture (WICSA'07)*, pages 4–4. Ieee, **2007**.

[31] Anton Jansen, Paris Avgeriou, and Jan Salvador van der Ven. Enriching software architecture documentation. *Journal of Systems and Software*, 82(8):1232–1248, **2009**.

[32] Andrew Forward and Timothy C Lethbridge. The relevance of software documentation, tools and technologies: a survey. In *Proceedings of the 2002 ACM symposium on Document engineering*, pages 26–33. **2002**.

[33] Philippe Kruchten, Patricia Lago, and Hans Van Vliet. Building up and reasoning about architectural knowledge. In *International conference on the quality of software architectures*, pages 43–58. Springer, **2006**.

[34] Anton Jansen, Jan Bosch, and Paris Avgeriou. Documenting after the fact: Recovering architectural design decisions. *Journal of Systems and Software*, 81(4):536–557, **2008**.

[35] Rick Kazman, Dennis Goldenson, Ira Monarch, William Nichols, and Giuseppe Valetto. Evaluating the effects of architectural documentation: A case study of a large scale open source project. *IEEE Transactions on Software Engineering*, 42(3):220–260, **2015**.

[36] Uwe van Heesch, Paris Avgeriou, and Antony Tang. Does decision documentation help junior designers rationalize their decisions? a comparative multiple-case study. *Journal of Systems and Software*, 86(6):1545–1565, **2013**.

[37] Jeff Tyree and Art Akerman. Architecture decisions: Demystifying architecture. *IEEE software*, 22(2):19–27, **2005**.

[38] Antony Tang, Paris Avgeriou, Anton Jansen, Rafael Capilla, and Muhammad Ali Babar. A comparative study of architecture knowledge management tools. *Journal of Systems and Software*, 83(3):352–370, **2010**.

[39] Dominik Rost, Matthias Naab, Crescencio Lima, and Christina von Flach Garcia Chavez. Software architecture documentation for developers: A survey. In *Software Architecture: 7th European Conference, ECSA 2013, Montpellier, France, July 1-5, 2013. Proceedings 7*, pages 72–88. Springer, **2013**.

[40] Sandun Dasanayake, Jouni Markkula, Sanja Aaramaa, and Markku Oivo. Software architecture decision-making practices and challenges: an industrial case study. In *2015 24th Australasian Software Engineering Conference*, pages 88–97. IEEE, **2015**.

[41] Emad Aghajani, Csaba Nagy, Mario Linares-Vásquez, Laura Moreno, Gabriele Bavota, Michele Lanza, and David C Shepherd. Software documentation: the practitioners' perspective. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, pages 590–601. **2020**.

[42] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Pearson Deutschland GmbH, **1995**.

[43] Robert C Martin. Clean architecture, **2017**.

[44] Paris Avgeriou and Uwe Zdun. Architectural patterns revisited-a pattern language. **2005**.

[45] F Buschmann, R Meunier, H Rohnert, P Sommerlad, and M Stal. A system of patterns: Pattern-oriented software architecture wiley new york. *Google Scholar Google Scholar Digital Library Digital Library*, **1996**.

[46] Brad Appleton. Patterns and software: Essential concepts and terminology. *Object Magazine Online*, 3(5):20–25, **1997**.

[47] Fatimah Mohammed Alghamdi and M Rizwan Jameel Qureshi. Impact of design patterns on software maintainability. *International Journal of Intelligent Systems and Applications*, 6(10):41, **2014**.

[48] Peter GW Keen and Michael S Scott Morton. Decision support systems: an organizational perspective. *(No Title)*, **1978**.

[49] Chiang Jao. *Decision support systems*. BoD–Books on Demand, **2010**.

[50] Gloria Phillips-Wren. Intelligent decision support systems. *Multicriteria decision aid and artificial intelligence: links, theory and applications*, pages 25–44, **2013**.

[51] Antony Tang, Maryam Razavian, Barbara Paech, and Tom-Michael Hesse. Human aspects in software architecture decision making: a literature review. In *2017 IEEE International Conference on Software Architecture (ICSA)*, pages 107–116. IEEE, **2017**.

[52] Rainer Weinreich, Iris Groher, and Cornelia Miesbauer. An expert survey on kinds, influence factors and documentation of design decisions in practice. *Future Generation Computer Systems*, 47:145–160, **2015**.

[53] Henry Muccini et al. Group decision-making in software architecture: A study on industrial practices. *Information and software technology*, 101:51–63, **2018**.

[54] Cornelia Miesbauer and Rainer Weinreich. Classification of design decisions–an expert survey in practice. In *Software Architecture: 7th European Conference, ECSA 2013, Montpellier, France, July 1-5, 2013. Proceedings 7*, pages 130–145. Springer, **2013**.

[55] Christine Hofmeister, Philippe Kruchten, Robert L Nord, Henk Obbink, Alexander Ran, and Pierre America. A general model of software architecture design derived from five industrial approaches. *Journal of Systems and Software*, 80(1):106–126, **2007**.

[56] Merve Ozdes Demir, Oumout Chouseinoglu, and Ayca Kolukısa Tarhan. Factors affecting architectural decision-making process and challenges in software projects: an industrial survey. *Journal of Software: Evolution and process*, **2024**.

[57] Colin Robson. Real world research: A resource for social scientists and practitioner-researchers. *(No Title)*, **2002**.

[58] Uwe Flick. *The SAGE handbook of qualitative data analysis*. Sage, **2013**.

[59] Michael Quinn Patton. *Qualitative research & evaluation methods: Integrating theory and practice*. Sage publications, **2014**.

[60] Clive Seale. Quality issues in qualitative inquiry. *Qualitative social work*, 1(1):97–110, **2002**.

[61] Philipp Mayring. Qualitative content analysis: Theoretical background and procedures. *Approaches to qualitative research in mathematics education: Examples of methodology and methods*, pages 365–380, **2015**.

[62] Beverley Hancock, Elizabeth Ockleford, and Kate Windridge. *An introduction to qualitative research*. Trent focus group London, **2001**.

[63] Lee J Cronbach. Coefficient alpha and the internal structure of tests. *psychometrika*, 16(3):297–334, **1951**.

[64] Darren George and Paul Mallery. *IBM SPSS statistics 26 step by step: A simple guide and reference*. Routledge, **2019**.

[65] Hsiu-Fang Hsieh and Sarah E Shannon. Three approaches to qualitative content analysis. *Qualitative health research*, 15(9):1277–1288, **2005**.

[66] Robert W Service. Book review: Corbin, j., & strauss, a.(2008). basics of qualitative research: Techniques and procedures for developing grounded theory . thousand oaks, ca: Sage. *Organizational Research Methods*, 12(3):614–617, **2009**.

[67] Barney Glaser and Anselm Strauss. *Discovery of grounded theory: Strategies for qualitative research*. Routledge, **2017**.

[68] Claes Wohlin, Martin Höst, and Kennet Henningsson. Empirical research methods in software engineering. *Empirical methods and studies in software engineering: Experiences from ESERNET*, pages 7–23, **2003**.

[69] David R Anderson, Dennis J Sweeney, Thomas A Williams, Jeffrey D Camm, and James J Cochran. *Statistics for business & economics*. Cengage Learning, **2016**.

[70] Klaas Andries de Graaf, Peng Liang, Antony Tang, and Hans van Vliet. How organisation of architecture documentation affects architectural knowledge retrieval. *Science of Computer Programming*, 121:75–99, **2016**.

[71] Iris Groher and Rainer Weinreich. A study on architectural decision-making in context. In *2015 12th Working IEEE/IFIP Conference on Software Architecture*, pages 11–20. IEEE, **2015**.

[72] Perla Velasco-Elizondo, Rosario Marín-Piña, Sodel Vazquez-Reyes, Arturo Mora-Soto, and Jezreel Mejia. Knowledge representation and information extraction for analysing architectural patterns. *Science of Computer Programming*, 121:176–189, **2016**.

[73] Mariem Haoues, Asma Sellami, Hanêne Ben-Abdallah, and Laila Cheikhi. A guideline for software architecture selection based on iso 25010 quality related characteristics. *International Journal of System Assurance Engineering and Management*, 8:886–909, **2017**.

[74] Ioanna Lytra, Carlos Carrillo, Rafael Capilla, and Uwe Zdun. Quality attributes use in architecture design decision methods: research and practice. *Computing*, 102:551–572, **2020**.

[75] Nancy R Mead and Ted Stehney. Security quality requirements engineering (square) methodology. *ACM SIGSOFT Software Engineering Notes*, 30(4):1–7, **2005**.

[76] Tingting Bi, Peng Liang, and Antony Tang. Architecture patterns, quality attributes, and design contexts: How developers design with them. In *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*, pages 49–58. IEEE, **2018**.

[77] Rick Kazman, Mark Klein, and Paul Clements. *ATAM: Method for architecture evaluation*. Carnegie Mellon University, Software Engineering Institute Pittsburgh, PA, **2000**.

[78] Robert L Nord, Mario R Barbacci, Paul Clements, Rick Kazman, Mark Klein, Liam O'Brien, and James E Tomayko. Integrating the architecture tradeoff analysis method (atam) with the cost benefit analysis method (cbam). *Sei-Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Institute*, pages 1–35, **2003**.

[79] Jeffrey Bell, Françoise Bellegarde, James Hook, Richard B Kieburtz, Alex Kotov, Jeffrey Lewis, Laura McKinney, DP Oliva, Tim Sheard, L Tong, et al. Software design for reliability and reuse: A proof-of-concept demonstration. In *Proceedings of the conference on TRI-Ada'94*, pages 396–404. **1994**.

[80] Maryam Razavian, Barbara Paech, and Antony Tang. Empirical research for software architecture decision making: An analysis. *Journal of Systems and Software*, 149:360–381, **2019**.

[81] Emma Bell, Alan Bryman, and Bill Harley. *Business research methods*. Oxford university press, **2022**.

[82] Antony Tang, Muhammad Ali Babar, Ian Gorton, and Jun Han. A survey of architecture design rationale. *Journal of systems and software*, 79(12):1792–1804, **2006**.

[83] Key architecture decisions template. `http://www.bredemeyer.com`, **2005**.

[84] Documenting architecture decisions. `http://thinkrelevance.com/blog/2011/11/15/documenting-architecture-`, **2011**.

[85] Atlassian software suite. `https://www.atlassian.com/software/confluence`, **2012**.

[86] Smrithi Rekha V and Henry Muccini. Suitability of software architecture decision making methods for group decisions. In *European Conference on Software Architecture*, pages 17–32. Springer, **2014**.

[87] Siamak Farshidi and Slinger Jansen. A decision support system for pattern-driven software architecture. In *European Conference on Software Architecture*, pages 68–81. Springer, **2020**.

[88] Martin Aruldoss, T Miranda Lakshmi, and V Prasanna Venkatesan. A survey on multi criteria decision making methods and its applications. *American Journal of Information Systems*, 1(1):31–43, **2013**.

[89] Omkarprasad S Vaidya and Sushil Kumar. Analytic hierarchy process: An overview of applications. *European Journal of operational research*, 169(1):1–29, **2006**.

[90] Rita A Ribeiro, Ana M Moreira, Pim Van den Broek, and Afonso Pimentel. Hybrid assessment method for software engineering decisions. *Decision Support Systems*, 51(1):208–219, **2011**.

[91] Rafael Capilla, Francisco Nava, Sandra Pérez, and Juan C Dueñas. A web-based tool for managing architectural design decisions. *ACM SIGSOFT software engineering notes*, 31(5):4–es, **2006**.

[92] Rafael Capilla, Francisco Nava, and Juan C Duenas. Modeling and documenting the evolution of architectural design decisions. In *Second Workshop on Sharing and Reusing Architectural Knowledge-Architecture, Rationale, and Design Intent (SHARK/ADI'07: ICSE Workshops 2007)*, pages 9–9. IEEE, **2007**.

[93] Rafael Capilla and Francisco Nava. Extending software architecting processes with decision-making activities. In *IFIP Central and East European Conference on Software Engineering Techniques*, pages 182–195. Springer, **2007**.

[94] Rafael Capilla, Francisco Nava, and Antony Tang. Attributes for characterizing the evolution of architectural design decisions. In *Third International IEEE Workshop on Software Evolvability 2007*, pages 15–22. IEEE, **2007**.

[95] Rafael Capilla, Francisco Nava, and Carlos Carrillo. Effort estimation in capturing architectural knowledge. In *2008 23rd IEEE/ACM International Conference on Automated Software Engineering*, pages 208–217. IEEE, **2008**.

[96] Rafael Capilla. Embedded design rationale in software architecture. In *2009 Joint Working IEEE/IFIP Conference on Software Architecture & European Conference on Software Architecture*, pages 305–308. IEEE, **2009**.

[97] Antony Tang, Yan Jin, and Jun Han. A rationale-based architecture model for design traceability and reasoning. *Journal of Systems and Software*, 80(6):918–934, **2007**.

[98] Anton Jansen, Tjaard de Vries, Paris Avgeriou, and Martijn van Veelen. Sharing the architectural knowledge of quantitative analysis. In *Quality of Software Architectures. Models and Architectures: 4th International Conference on the Quality of Software-Architectures, QoSA 2008, Karlsruhe, Germany, October 14-17, 2008. Proceedings 4*, pages 220–234. Springer, **2008**.

[99] Sol Jin, Xu Lian, Hanearl Jung, Jinsoo Park, and Jihae Suh. Building a deep learning-based qa system from a cqa dataset. *Decision Support Systems*, page 114038, **2023**.

[100] Ahmad Abdellatif, Khaled Badran, and Emad Shihab. Msrbot: Using bots to answer questions from software repositories. *Empirical Software Engineering*, 25:1834–1863, **2020**.

[101] Johnathan Mauricio Calle Gallego and Carlos Mario Zapata Jaramillo. Quare: towards a question-answering model for requirements elicitation. *Automated Software Engineering*, 30(2):25, **2023**.

[102] Xiaoli Lian, Wenchuang Liu, and Li Zhang. Assisting engineers extracting requirements on components from domain documents. *Information and Software Technology*, 118:106196, **2020**.

[103] Open ai models. `https://platform.openai.com/docs/models/gpt-3-5-turbo`.

[104] Atlassian. `https://www.atlassian.com/software/jira`.

[105] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5):513–523, **1988**.

[106] Chuan Wan, Yuling Wang, Yaoze Liu, Jinchao Ji, and Guozhong Feng. Composite feature extraction and selection for text classification. *IEEE Access*, 7:35208–35219, **2019**.

[107] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, **2018**.

[108] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, **1997**.

[109] Leo Breiman. Random forests. *Machine learning*, 45:5–32, **2001**.

[110] Marti A. Hearst, Susan T Dumais, Edgar Osuna, John Platt, and Bernhard Scholkopf. Support vector machines. *IEEE Intelligent Systems and their applications*, 13(4):18–28, **1998**.

[111] Leif E Peterson. K-nearest neighbor. *Scholarpedia*, 4(2):1883, **2009**.

[112] Tianqi Chen, Tong He, Michael Benesty, Vadim Khotilovich, Yuan Tang, Hyunsu Cho, Kailong Chen, Rory Mitchell, Ignacio Cano, Tianyi Zhou, et al. Xgboost: extreme gradient boosting. *R package version 0.4-2*, 1(4):1–4, **2015**.

[113] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30, **2017**.

[114] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, **2011**.

[115] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, **2017**.

[116] Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In

Proceedings of the IEEE international conference on computer vision, pages 19–27. **2015**.

[117] Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. How to fine-tune bert for text classification? In *Chinese Computational Linguistics: 18th China National Conference, CCL 2019, Kunming, China, October 18–20, 2019, Proceedings 18*, pages 194–206. Springer, **2019**.

[118] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: Smaller, faster, cheaper and lighter. arxiv 2019. *arXiv preprint arXiv:1910.01108*, **2019**.

[119] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, **2019**.

[120] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27, **2014**.

[121] Duyu Tang, Bing Qin, and Ting Liu. Document modeling with gated recurrent neural network for sentiment classification. In *Proceedings of the 2015 conference on empirical methods in natural language processing*, pages 1422–1432. **2015**.

[122] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144. **2016**.

[123] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. *Advances in neural information processing systems*, 30, **2017**.

[124] Yuming Li, Johnny Chan, Gabrielle Peko, and David Sundaram. An explanation framework and method for ai-based text emotion analysis and visualisation. *Decision Support Systems*, 178:114121, **2024**.

[125] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, **2015**.

[126] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794. **2016**.

[127] Andy Liaw, Matthew Wiener, et al. Classification and regression by randomforest. *R news*, 2(3):18–22, **2002**.

[128] Robi Polikar. Ensemble learning. *Ensemble machine learning: Methods and applications*, pages 1–34, **2012**.

[129] Pablo N Mendes, Max Jakob, Andrés García-Silva, and Christian Bizer. Dbpedia spotlight: shedding light on the web of documents. In *Proceedings of the 7th international conference on semantic systems*, pages 1–8. **2011**.

[130] Spacy dbpedia spotlight. `https://pypi.org/project/spacy-dbpedia-spotlight/`.

[131] Alexander Shvets. Dive into design patterns. *Refactoring. Guru*, **2018**.

[132] Orestis Lampridis, Laura State, Riccardo Guidotti, and Salvatore Ruggieri. Explaining short text classification with diverse synthetic exemplars and counter-exemplars. *Machine Learning*, 112(11):4289–4322, **2023**.

[133] Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Dino Pedreschi, Franco Turini, and Fosca Giannotti. Local rule-based explanations of black box decision systems. *arXiv preprint arXiv:1805.10820*, **2018**.

APPENDICES

Appendix A. Validation with Stack Overflow Data

The Table 6.1 compares the system's recommendations against accepted answers from Stack Overflow for selected queries tagged with 'software-design' and 'design-patterns'. Out of the 20 queries evaluated, SOFAR-DSS's recommendations aligned with the accepted answers in 17 instances, demonstrating an 85% alignment rate. This highlights the system's ability to provide viable solutions that are consistent with community-endorsed practices.

Table 6.1 Design Pattern Recommendations Based on Stack Overflow Queries

Stack Overflow Query (URL)	**Answer**	**Response of SOFAR-DSS**
https://tinyurl.com/mrncha4w	Implement yourself a Factory. Your factory will produce a list of available IReports and possibly return some metadata about them.	Factory method, template method
https://tinyurl.com/yrd6muck	Your solution probably will use the Iterator pattern, and could also use the Adapter (or wrapper) pattern.	Adapter, MVC, Iterator
https://tinyurl.com/yc4xnn9b	This sounds like you need to use the factory pattern	Factory method, template method
https://tinyurl.com/3256m3aj	Most likely a Flyweight is what you are looking for.	Object pool, flyweight
		Continued on next page

Table 6.1 (continued)

Stack Overflow Query (URL)	**Answer**	**Response of SOFAR-DSS**
https://tinyurl.com/2wwucfdu	I'll throw strategy out there as well, but we may be getting off-track.	Adapter, Builder, State
https://tinyurl.com/4p8bkz7h	Strategy is basically configuring your system at runtime with a general, consistent way to handle some aspect of the operation, in this case generating the requests.	Command, abstract factory
https://tinyurl.com/yc4xnn9b https://tinyurl.com/37vmhchj	This sounds like you need to use the factory pattern One other approach, might be for your framework to handle what it does, in an abstract and common way, through your own set of classes.	Command, strategy, Bridge Adapter, Model view adapter,strategy
https://tinyurl.com/vb5znnbj	You need to want to use an abstract factor as you seem to have high interaction multiple instances of different source system interfaces APIs.	Builder, adapter, decorator

Continued on next page

Table 6.1 (continued)

Stack Overflow Query (URL)	**Answer**	**Response of SOFAR-DSS**
https://tinyurl.com/3d5byyap	Sounds like a clear cut case for the Observer Pattern, whereby Newspaper and Document will be Subjects and your observers will be Observers.	Factory, observer, singleton
https://tinyurl.com/37vmhchj	Indeed, factory would be quite reasonable, but combined with the Strategy pattern (as noticed in the other answers).	Command, strategy, Bridge
https://tinyurl.com/3xuv5j6f	One other approach, might be for your framework to handle what it does, in an abstract and common way, through your own set of classes. And then have two different strategies for rendering the content (two separate renderers, one for Word and one for PowerPoint).	Adapter, Model view adapter,strategy

Continued on next page

Table 6.1 (continued)

Stack Overflow Query (URL)	**Answer**	**Response of SOFAR-DSS**
https://tinyurl.com/2xrbajnr	Strategy Pattern: It will define the notification strategy based on the contexts like email, push, WhatsApp, etc.	Template, Strategy, Observer
https://tinyurl.com/mw4vzyk8	Short Answer: Monitor design pattern.	Observer, Command
https://tinyurl.com/46zp2dhe	As you need to communicate to different sources for input, it would be good to have that part completely asynchronous so that you don't block your main program for that.	Builder, adapter, decorator
https://tinyurl.com/uaf5wvmp	Sounds like you want the Adapter pattern.	Strategy, adapter
https://tinyurl.com/232awp5x	I suggest you to implement a Factory Method and let Symfony instantiate the correct class for you (as described here).	Factory, adapter, mvc

Continued on next page

Table 6.1 (continued)

Stack Overflow Query (URL)	**Answer**	**Response of SOFAR-DSS**
https://tinyurl.com/3d5byyap	You could think using the observer pattern to monitor the event you are interested at and notify appropriately a list of observers when changes occur so they may respond accordingly.	Factory, observer, singleton
https://tinyurl.com/4msw6zt6	Sounds like a clear cut case for the Observer Pattern, whereby Newspaper and Document will be Subjects and your Notifications will be Observers.	State, strategy, observer
https://tinyurl.com/a76j6e72	If something reflect ”is” relationship use inheritance, if something reflect ”has” relationship use aggregation.	Factory, Strategy
https://tinyurl.com/5y6smtcr	Ignoring the fact that you have missed the No.1 Golden Rule of coding, namely that everything should be named correctly, you could use the Iterator and Composite patterns.	Strategy design pattern, Composite design pattern
		Continued on next page

Table 6.1 (continued)

Stack Overflow Query (URL)	Answer	Response of SOFAR-DSS
https://tinyurl.com/kcrx2psa	Iterator is used to traverse a container and access the container's elements. The iterator pattern decouples algorithms from containers.	Observer, Factory method pattern

Appendix B. Validation with Design Pattern Cases from Book

In this appendix, we provide details of the design patterns discussed in the main text using specific cases taken from a book. This validation aims to illustrate the practical applicability of these patterns in addressing common software development challenges.

Table 6.2 Book Cases with Their Design Patterns and the SOFAR-DSS's Responses

Problem	Solution	Response of SOFAR-DSS
An application needs a single database connection to ensure data integrity and optimize resource usage.	Singleton	Singleton, Abstract Factory
A text editor needs to provide undo and redo functionality.	Memento	Command-line interface, command pattern
In a gaming application, the behavior of a player character needs to change depending on the character's state (like standing, jumping, running).	State	State, Strategy

Table 6.2 Design Pattern Recommendations (continued)

Problem	**Solution**	**Response of SOFAR-DSS**
An application needs to create the elements of a complex aggregate. The specification for the aggregate exists on secondary storage and one of many representations needs to be built in primary storage.	Builder	Builder, Abstract Factory
A framework needs to standardize the architectural model for a range of applications, but allow for individual applications to define their own domain objects and provide for their instantiation.	Factory	Abstract Factory, Factory
Application ”hard wires” the class of object to create in each ”new” expression.	Prototype Design	Abstract Factory, Factory
Application needs one, and only one, instance of an object. Additionally, lazy initialization and global access are necessary.	Singleton	Singleton, Factory method

Table 6.2 Design Pattern Recommendations (continued)

Problem	**Solution**	**Response of SOFAR-DSS**
An ”off the shelf” component offers compelling functionality that you would like to reuse, but its ”view of the world” is not compatible with the philosophy and architecture of the system currently being developed.	Adapter	MVC, Adapter
Hardening of the software arteries” has occurred by using subclassing of an abstract base class to provide alternative implementations. This locks in compile-time binding between interface and implementation. The abstraction and implementation cannot be independently extended or composed.	Bridge	Strategy, Abstract Factory
Processing of a primitive object is handled one way, and processing of a composite object is handled differently. Having to query the ”type” of each object before attempting to process it is not desirable.	Composite	Composite, Visitor

Table 6.2 Design Pattern Recommendations (continued)

Problem	**Solution**	**Response of SOFAR-DSS**
You want to add behavior or state to individual objects at run-time. Inheritance is not feasible because it is static and applies to an entire class	Decorator	Strategy, Decorator
A segment of the client community needs a simplified interface to the overall functionality of a complex subsystem.	Facade	Facade, Adapter
You need to support resource-hungry objects, and you do not want to instantiate such objects unless and until they are actually requested by the client.	Proxy	Flyweight, Proxy

Table 6.2 Design Pattern Recommendations (continued)

Problem	**Solution**	**Response of SOFAR-DSS**
If an application is to be portable, it needs to encapsulate platform dependencies. These ”platforms” might include: windowing system, operating system, database, etc. Too often, this encapsulation is not engineered in advance, and lots of #ifdef case statements with options for all currently supported platforms begin to procreate like rabbits throughout the code.	Abstract Factory	Factory, Abstract Factory
Designing objects down to the lowest levels of system ”granularity” provides optimal flexibility, but can be unacceptably expensive in terms of performance and memory usage.	Flyweight	Flyweight, Composite
A class of requests occurs repeatedly in a well-defined and well-understood domain. If the domain were characterized with a ”language”, then problems could be easily solved with an interpretation ”engine”.	Command	Command, Mediator