International Journal of Human Capital and Information Technology Professionals
Volume 15 • Issue 1

Collaborative Solutions to Software Architecture Challenges Faced by IT Professionals

Zeeshan Anwar, National University of Sciences and Technology, Pakistan*

https://orcid.org/0000-0002-8029-0604

Nazia Bibi, National University of Sciences and Technology, Pakistan

Tauseef Rana, National University of Sciences and Technology, Pakistan

Seifedine Kadry, Department of Applied Data Science, Noroff University College, Kristiansand, Norway & Artificial Intelligence Research Center (AIRC), Ajman University, Ajman, UAE & Department of Electrical and Computer Engineering, Lebanese American University, Byblos, Lebanon & MEU Research Unit, Middle East University, Amman, Jordan

Hammad Afzal, National University of Sciences and Technology, Pakistan

ABSTRACT

Software architecture serves as a crucial link between problem and solution domains in software systems. However, reliance on graphical artifacts for architecture design has limitations, especially in abstract analysis. To overcome these constraints, Architecture Description Languages (ADLs) offer a more formal approach. Yet, our research reveals that ADLs face numerous challenges, as identified through interviews, surveys, and community interactions. By mining content from various sources, including mailing lists and forums, we comprehensively explore the concerns of software engineers. Employing content mining, topic modeling, and grounded theory, we compile a database of real-world issues and solutions in software architecture. Comparing our findings with existing literature, we identify 17 primary issues faced by practitioners. We also compare our results with language models to ascertain areas of agreement and disagreement. Finally, we propose solutions for each identified issue to aid future analysts.

KEYWORDS

Architecture Description, Community Question Answering, Content Mining, IT Professionals, Practical Knowledge, Software Architecture

1. INTRODUCTION

Perry and Wolf introduced the significance and foundation of software architecture (Taylor et al., 2021). A software system's architecture is the system's structure in terms of the program units (components) with externally visible proper- ties and connections between these components (Bass et al., 2003; Hasselbring, 2018). Semantically, the high-level design of a system bridges the gap between the

DOI: 10.4018/IJHCITP.342839 *Corresponding Author

problem space (requirements) and the solution space (implementation). The architecture of a system can be represented in different ways that are (Land, 2002; Malavolta et al., 2013; Hasselbring, 2018): (i) informally by utilizing boxes-and-lines, (ii) semi-formally by utilizing a modeling language (e.g., UML) (Deryugina et al., 2019) and (iii) formally by utilizing an Architectural Description Language (ADL) (Fuxman, 2000).

For software systems, architecture dependent on boxes and lines has constrained expressiveness-ability; henceforth, it's not valuable for in-depth analysis and documentation. Interestingly, a formal portrayal dependent on an ADL includes further subtle elements and depictions of the design, which can be helpful for various purposes (e.g., better dependency and consistency analyses) (Taylor, 2019). Despite its promising highlights, ADLs are not generally utilized in industry (Fuxman, 2000) due to the trouble of utilization and the absence of tool support. The primary explanations behind this are ADLs are created in confinement (Garlan et al., 2010), ADLs are not general purpose (Woods and Bashroush, 2012), have no standard (Mishra and Dutt, 2005), produced for scholastic viewpoint (Fuxman, 2000) and communication gap among researchers and practitioners (Bradbury et al., 2004). Due to convenience, UML-based representation is being utilized at a wider scale in the industry (Fuxman, 2000; Taylor, 2019). In the software component model survey (OMG, 2007), UML is considered an ADL. Rather than boxes-and-lines portrayal, ADL (and UML) based representation is better for human and machine readability. But UML-based representation does not solve structural and technical issues (Taylor, 2019).

In the existing work, surveys are used to categorize the architecture representations and identify the architecture- related issues. For example, surveys carried out to explore and categorize the features of ADL based representations (Bass et al., 2003; Land, 2002; OMG, 2007; Fuxman, 2000; Kamal and Avgeriou, 2007; Khan et al., 2016; Capilla et al., 2016; Shahin et al., 2014) and identification of architecture related issues (Tian et al., 2019; Tamburri et al., 2019; Othmane and Lamm, 2019; Cai and Kazman, 2019). All these studies are a sort of specialized work as these include findings depending on some pre-defined criteria/ framework. For example, the work in (OMG, 2007; Kamal and Avgeriou, 2007) is carried out for a limited set of ADLs. Tian et al. (Tian et al., 2019) applied grounded theory on 207 Stack Overflow posts to extract architecture smells. In another paper, Tamburri et al. (Tamburri et al., 2019) surveyed and identified 10 architecture smells in an agile environment. A tool DV8 (Cai and Kazman, 2019) was developed to detect architecture anti-patterns and measure modularity. Similarly, Othmane et al. (Othmane and Lamm, 2019) found that architects do not properly document architecture; therefore, a tool based on a gamification mechanism was proposed.

Although closely related to software architecture, the studies mentioned above lack collecting the stakeholders' voices and are limited to a single topic. As identified by Taylor (Taylor, 2019), the future software architecture concerns are modeling, knowledge capture, Evolution, and Ecosystems. There is a need to conduct a large-scale study that covers the various aspects of software architecture. To fill this gap, we conducted this research. Our primary objective is to conduct a large-scale study that encompasses the diverse dimensions of software architecture, thereby capturing the multifaceted challenges and considerations faced by stakeholders in the field. By leveraging data sourced from a myriad of community sites and expanding our scope to encompass a wide array of software architecture topics, we aim to address the lacuna identified in prior research efforts. Through this holistic approach, we seek to provide a nuanced understanding of the evolving landscape of software architecture, shedding light on critical areas that warrant attention and offering insights that can inform future developments and initiatives in the field.

Utilizing search queries/keywords *Architecture Description Language(s)*, *Software Architecture Tools* and *Software Architecture*, we gathered information from community question-answering (CQA) sites, mailing lists, and forums such as stack exchange network, yahoo answers, quora, etc. Consequently, the extent of our work is broader than the related work. For gathering and examining the extensive informational collection of the previously mentioned sources, our methodology can be considered the big data analytic to study software architecture and languages/tools to support designing

software architecture. Our research methodology is more robust as compared with existing research as it is a mixture of qualitative and quantitative methods. We analyzed the data using text analytics, network analysis, topic modeling, and grounded theory. This research aims to gather and examine the concerns of software engineers and IT Professional while utilizing software architecture. The results of the proposed model are validated with the published literature and Large Language Models (LLMs) GPT and BRAD. The area of agreement and disagreement between the proposed model and LLMs are reported to shows the strength of the proposed model in correctly identifying the concerns of the IT Professionals. The major contributions of our research are given below:

- Our work contributes a comprehensive database of software architecture issues and corresponding solutions, avail- able for access on GitHub[1]. This resource serves as a valuable repository for practitioners and researchers, offering insights into prevalent challenges and potential remedies in software architecture.
- Our research prioritizes and categorizes the major issues encountered in software architecture, shedding light on the most pressing challenges faced by practitioners. By organizing these issues into distinct categories, we aim to provide a structured framework for understanding and addressing architectural complexities.
- Our analysis reveals the interrelatedness of issue categories in software architecture and facilitates the development of grounded theories based on identified challenges. By uncovering underlying patterns and relationships, we contribute to a deeper understanding of architectural dynamics and inform evidence-based decision-making.
- We have made our research materials, including project data and scripts, openly accessible on GitHub, promoting transparency and reproducibility in research. By sharing these resources, we aim to facilitate collaboration and knowledge dissemination within the software architecture community.

The rest of the paper's literature review is given in Section 2. Our methodology is given in Section 3, whereas analysis and results are given in Section 4. A comparison of results with literature is given in 5. Finally, Section 6 presents the conclusion and future work.

2. LITERATURE REVIEW

This is a multi-disciplinary paper, i.e., a blend of software architecture knowledge, text mining strategies, and mixed methods. Along these lines, we subdivided the literature review into three subsections. Section A clarifies different ideas about software architecture and strategies to draw architecture. Different review papers about software architecture are given in section B. Related work on mining software engineering repositories is given in section C.

2.1 Software Architecture

Generally, architecture is represented by informal lines and circle drawings in which the components, properties, connections, and system behavior are defined poorly (Bass et al., 2003). Architecture patterns and styles are two terminologies used in the system domain, but these terms come from two different schools of thought. Architectural patterns specify problem-solution space; for every problem, a generally proven solution can address the problem. Whereas architectural styles represent the rules that identify the components and connectors to connect these components (Kamal and Avgeriou, 2007).

In architecture design, creating software architecture and adding a description transforms the architectural information into a viable model. Based on the architecture that is created with considerations in changing requirements from the market, technology advancement, architecture dependencies, and other possible factors, a road map for engineering will provide guidance on what

feature or software component will be implemented first and which will be implemented next, or later on in the different versions of a software system (Zhao, 2016).

The architectural design is very important to determine the success of software. Therefore, there must be a tool to support architectural design that can check the consistency and correctness of the architecture. For this purpose, many ADLs are developed (to support software architecture) in isolation. Hence, different ADLs have different capabilities and are mostly domain-specific; this leads to the disadvantage of combining features of different ADLs. To address this disadvantage, some examples of attempts are mentioned in the rest of this section. Acme solved this problem by combining features of different ADLs in Acme Studio. The authors explain the features of Acme and combine the features of Wright and Rapide to develop architecture (Garlan et al., 2010). Navasa et al. (Navasa et al., 2009) present a new ADL named DAOP-ADL (Dynamic Aspect-Oriented Platform-ADL) by focusing on component-based software engineering and aspect-oriented programming. In another study (Mckenzie et al., 2004), two ADLs, Rapide, and Acme, are used for separate purposes. These experiments prove the usefulness of ADLs for the design and analysis of enterprise architectures. One reason ADLs are not widely used is the communication gap between the practitioners and the research community.

Many of the developed ADLs are domain-specific and do not meet the needs of the architecture of general systems. Wide varieties of ADLs are available in the research domain, but these are seldom applied to the real information system (Malavolta et al., 2013).

Effective software architecture and design practices are crucial for the success of projects across the software life cycle. Work of Whiting et al. (Whiting and Andrews, 2020), explores the significance of maintaining a stable architecture, identifies symptoms of architecture drift and erosion, and examines existing tools and methods to mitigate these challenges.

In today's dynamic software landscape, systems must continually evolve and adapt to remain competitive. This paper addresses the need for a systematic approach to software architecture reconstruction, aiming to identify common activities and elements essential for guiding this process effectively. Through rigorous research including a systematic literature review and survey, we propose a process termed Software Improvement in the Reconstruction of Architectures (SIRA). SIRA integrates and extends previous research, offering a structured framework for semi-automated software architecture reconstruction. Additionally, this study identifies key components of the reconstruction process, including techniques, architectural elements, and automation tasks (Guamán et al., 2020).

2.2 Software Architecture Surveys

In this section, various survey papers related to software architecture are reviewed. A comparison of these papers is given in Table 1. Software architecture and design are the main pillars of software engineering. A buggy software may cause problems; therefore, a design should be tested and analyzed early. This can reduce the risks associated with the development and maintenance cost. Software architecture maintains the conceptual integrity of the system. In his research, Taylor reviewed both domain-specific and domain-independent software architecture techniques. The need and evaluation of various tools and techniques are also discussed. ADLs were created to represent architecture and solve structural and technical issues, but the business context and domain-specific concerns cannot be modeled using ADLs. UML is good for modeling business context and components of architecture. As described by Taylor, analyzing the architecture requires additional cost because a formal model is required for analysis; therefore, the value of the information produced by analysis must be considered before running the analysis. As identified by the author, the future concerns of software architecture are Modeling, knowledge capture, Evolution, and Ecosystems (Taylor, 2019).

Architecture smells are given less attention as compared with code smells. In his research, Tian et al. (Tian et al., 2019) extracted 207 posts related to architecture smell from stack overflow and applied grounded theory to analyze the extracted posts. MAXQDA tool is used for grounded theory. As a

Table 1. Analysis of Surveys

Paper	Scope	Limitations
(Taylor, 2019)	Reviewed domain-specific and domain-independent tools for architecture. Strengths & Weaknesses of UML and ADLs are given. The future concerns related to architecture are identified.	This paper is based on the review of existing literature.
(Tian et al., 2019)	In this paper, architecture smells are analyzed.	Only stack overflow posts related to architecture smells are analyzed.
(Hasselbring, 2018)	In this paper, authors studied architecture's past, present, and future. In the future, highly modular architecture will be required for agile development. The emphasis of this paper is on the architecture for agile development.	
(Clements, 1996)	A survey of ADLs is made to discover their features and system support.	Only limited to ADLs.
(Khan et al., 2016)	Author conducted a survey to deal with NFR at the architecture level because NFR is dealt with at later stages of development.	This work is limited to dealing with NFR at the architecture level.
(Keeling, 2015)	Four trends of architecture are: Architecting for DevOps, Flexible Designs, Lightweight Architecture Design Methods, and Renewed Interest in Software Architecture Fundamentals.	An overview of conference papers is given by the authors.
(Mayer and Weinreich, 2019)	Data from 30 respondents is selected to analyzed and co-relate community smells and architecture smells.	Only architecture and community smells are analyzed in an agile environment.
(Tamburri et al., 2019)	Investigates perspectives of key figures in software architecture research on empirical methodologies, highlighting preferences and challenges.	Limited to perceptions within the soft- ware architecture community, may not fully represent broader views in soft- ware engineering.
(Galster and Weyns, 2023)	Investigating the textual representation and prevalence of architectural knowledge concepts in issue trackers like Jira, with implications for enhancing software development practices.	Limited to analysis of issues from three Apache projects, may not fully capture the diversity of architectural knowledge concepts across different software development contexts.
(Soliman et al., 2021)	Related to documenting architecture.	

result, various architectural smells, tools for dealing with architectural smells, causes of architectural smells, and quality attributes affected by architectural smells are identified.

Wilhelm Hasselbring surveyed the software architectures past, present, and future. In the past, software architecture was represented using box-and-line diagrams and UML. But these notations are informal, therefore, formal models like ADLs were created. Software product lines have also become popular for reusing components. Presently, the focus of the architecture community is on micro services and quality requirements. Services are used to develop the system quickly in multiple deployable

units. Quality requirements like performance, modifiability, and security can be implemented at the architecture level. The future research on architecture is its integration into the agile framework because architecture once developed is hard to change in agile changing environment. To solve this, highly modular architecture is required, and architecture is enabled in all phases of agile (Hasselbring, 2018).

Architecture Description Languages (ADLs) formally represent a system's architecture. By increasing in number, they differ in abstraction and analysis capabilities. In (Clements, 1996), the authors summarize a survey of ADL's that characterizes ADLs in terms of (a) the classes of systems they support, (b) the inherent properties of the languages themselves, and (c) the support of process and technology they provide to represent, refine, analyze, and build systems from an architecture.

Roohullah et al. surveyed how to deal with non-functional requirements (NFR) at the architectural level. The authors emphasize the integration of non-functional requirements and architecture. It is very important to address NFRs as part of the architecture. Functional requirements are considered at the initial stages of a software project by neglecting the NFRs. NFRs are usually handled in the final stages of the project, which means quality compromise. NFRs and architecture complement each other, so both are treated at the architectural level. Run time and not run time NFRs should both be catered to at the architectural level (Khan et al., 2016).

SATURN-14 conference targeted four trends: Architecting for DevOps, Flexible Designs, Lightweight Architecture Design Methods, and Renewed Interest in Software Architecture Fundamentals. DevOps is basically a growing software development approach that covers both traditional operations as well as to increase software development to enhance origination's ability to deliver business value. Secondly, the purpose of this conference is to target how to achieve design flexibility so that it is flexible enough to accommodate changes. The concept of lightweight architecture is also discussed. Trimming fats from architecture is always a good idea, i.e., architecture is lightweight. Architectural modeling skill, lightweight representations, and discussion of the lean design method is part of the SATURN technical program (Keeling, 2015).

Both communities of researchers and practitioners understand the value of Architecture Knowledge (AK) but this knowledge evaporates with time if not documented. Lack of motivation is the major reason for not documenting AK. Authors used gamification to capture the AK as gamification motivates the users. Software Architecture Knowledge Management (SAKM) toolkit was created which document architecture profile and provide gamification mechanism. A focused group study was conducted with graduate students but authors did not find any statically significant evidence about effect of gamification on capturing AK. But interview with students show that gamification is worthwhile for AK capturing (Mayer and Weinreich, 2019).

These days most of the companies are using agile methods for software development. Agile methods in distributed teams may develop community smells like time wrap, cognitive distance, newbie free riding, power distance etc. These community smells effect the quality of software being developed. Authors analyzed 30 software organization and co- relate community smells and architecture smells. Survey, Delphi and interviews are used for data collection. Data from 30 respondents is selected for analysis. Ten architecture smells e.g. sloppy modularization, untraceable business requirement, impossible component swap etc. are identified by authors in their analysis. These architecture smells are correlated with community smells (Tamburri et al., 2019).

Research in software engineering has raised concerns about empirical studies, including reproducibility and the relevance of findings. However, little is known about how these concerns are viewed by researchers and evaluators. Focusing on software architecture, a subfield of software engineering, this study investigates the perspectives of 105 key figures in architecture research on empirical research methods. Findings reveal a lack of consensus on preferences for quantitative versus qualitative methods, the value of professional versus student participants, and the prioritization of internal versus external validity. While replication studies are generally valued, challenges in execution are acknowledged. These findings highlight the limited consensus on empirical research

Table 2. Challenges faced by IT Professionals in Software Architecture Design

Article	Identified Challenges
(Rehman and Khan, 2022)	Lack of common development, reliability, management issues, and development limitations.
(Di Pompeo and Tucci, 2023)	Estimation and improvement of quality attributes is challenging and time-consuming. Improvement of quality attributes may require contrasting refactoring actions on the architecture.
(Ho-Quang et al., 2020)	Little research in software architecture validated beyond individual case studies. Huge effort needed for Big Data-driven research.
(Villegas et al., 2017)	Separation of concerns between monitoring, self-adaptation controller, and control objectives.
(Seifermann et al., 2018)	Approaches for maintaining security properties fail to exploit architectural design.

methodologies within the software architecture community, with implications for researcher training and practice (Galster and Weyns, 2023).

The research outlines a study investigating how architectural knowledge concepts are conveyed in issue trackers like Jira, examining their textual representation, prevalence, and relationships. Analyzing issues from three Apache projects, the study identifies architecturally relevant items by linking them to pertinent source code changes. Through manual la- beling and coding, the research aims to facilitate the development of effective search tools for identifying architectural knowledge concepts in issues, potentially enhancing software development practices. While the abstract provides a succinct overview of the research aims and methodology, further clarity on specific findings and practical implications would enhance its comprehensiveness (Soliman et al., 2021).

The research of Wan et al. (Wan et al., 2023) study the gap in understanding how practitioners engage in software architecture-related activities and the challenges they encounter. To address this gap, authors conducted interviews with 32 practitioners representing 21 organizations across three continents. Through this research, authors identified the key challenges faced by practitioners in software architecture practice throughout various stages of software development and maintenance. This study highlights common activities in software architecture, spanning requirements analysis, de- sign, construction, testing, and maintenance, and elucidates the corresponding challenges faced by practitioners. Notably, findings reveal that many of these challenges revolve around issues related to management, documentation, tooling, and processes. Furthermore, authors compile recommendations to mitigate these challenges, offering valuable insights for practitioners and researchers alike in addressing the complexities of software architecture practice.

2.3 Issues Faced by IT Professionals in Software Architecture Design

Common challenges are encountered in software architecture design, including issues related to development consistency, architectural quality, reliability, management effectiveness, environmental factors, development constraints, cost considerations, skill shortages in highly specialized workforce, technology adequacy, traditional co-location models, privacy concerns, communication workflow gaps, trust deficits, market expansion hurdles, framework integration complexities, effectiveness limitations, assessment deficiencies, and scheduling inefficiencies (Rehman and Khan, 2022). Moreover, the estimation and enhancement of quality attributes in software architectures pose considerable challenges, demanding significant time and effort (Di Pompeo and Tucci, 2023). Similarly, the design of data-intensive systems presents its own set of obstacles, encompassing concerns such as

data management, system design and implementation challenges, messaging service complexities, security and privacy apprehensions, data volume issues, data dissemination hurdles, data curation challenges, considerations related to software reuse and the utilization of open-source software, data search intricacies, data processing and analysis complexities, and the intricacies of information modeling (Dimov et al., 2022). A list of challenges identified in various articles are given in Table 2.

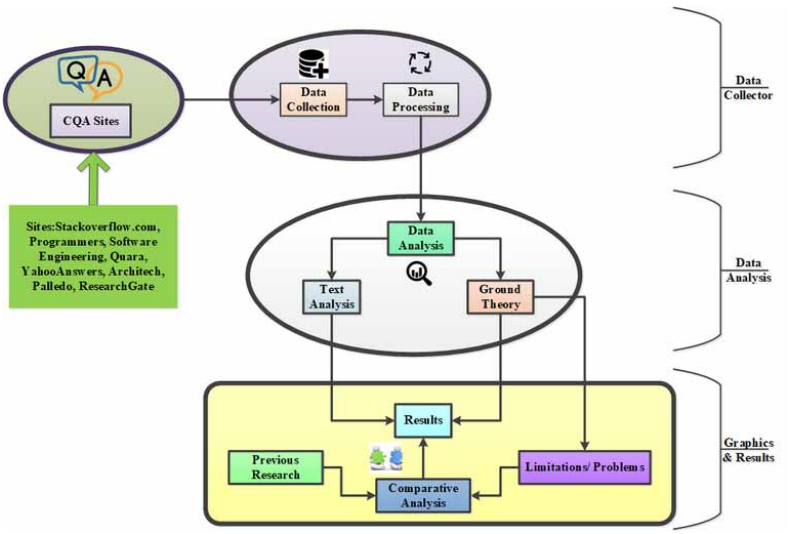
2.4 Mining Software Repositories

There are hundreds of tools and techniques for analyzing and mining the data. In this section we limit our focus to analysis tools that are used for analyzing the community question answering (CQA) sites only. The main application of these tools is text mining of CQA sites. The analysis tools used in literature are for text features extractions (text mining and topic modelling), data visualization and analysis, textual similarity, and classification and prediction. Some of the literature in which Stack Overflow (SO), a CQA site, data set is used for aforementioned tasks is given below:

Using Latent Dirichlet Allocation (LDA) for topic identification, Allamanis et al shows that confusing programming concepts can be identified by analysing SO data (Allamanis and Sutton, 2013). In (Arwan et al., 2015), source code is retrieved from SO data (by using Mallet and LDA) to facilitate programmers. Barua et al (Barua et al., 2014) created a topic set after applying (by use of Mallet) on SO data. Another very interesting study (Campbell et al., 2013) identifies deficient project documentation by applying LDA on project documentation and SO questions. In (Bazelli et al., 2013), the personality traits of stack overflow users are studied by using Linguistic Inquiry and Word Count (LIWC). In (Bosu et al., 2013), authors proposed guideline for users who want to gain reputation quickly on SO. Souza et al (de Souza et al., 2014) used textual similarity and crowd evaluation mercies to design a recommendation system to assist development. Anwar et al. (Anwar et al., 2023) developed model based on the CNN and LSTM to categorize the Stack Overflow question based on the content quality. In another research, Anwar et al. proposed a multi-model based framework to use Stack Overflow data for open innovation in software engineering (Anwar and Afzal, 2024).

Research methodology of Bajaj et al. (Bajaj et al., 2014) is somehow relevant to our work. Bajaj had used LDA and Natural Language Processing (NLP) toolkit for mining SO data after using a stemming algorithm. The authors use mixed method analysis. Topics are first categorized then hot

Figure 1. Research Methodology



topics are identified, temporal trends are identified, web topics in mobile development are identified and main technical challenges are identified.

3. RESEARCH METHODOLOGY

Rather than the ordinary methodologies in the related work, our methodology is novel that gathers the genuine questions with respect to software architecture posed by the practitioners. The common steps required to analyze information from web are: 1) data extraction, 2) data pre-processing, 3) data analysis and 4) displaying results. Our philosophy for gathering and examining information is given in Figure 1. The major modules of research methodology are given in Figure 2 and are described in the subsequent sections.

3.1 Data Collection

The data collection process involved selecting a diverse range of social learning platforms, including blogs, forums, community question answering (CQA) sites, and mailing lists, renowned for their widespread popularity and relevance to the domain of software architecture. Our selection criteria were carefully crafted to prioritize platforms based on their user engagement, dataset size, and alignment with software architecture themes. Our data collection efforts commenced in April 2021 and updated in June 2023, to incorporate the latest data in the research.

To construct a comprehensive dataset comprising questions and answers, we focused on eight prominent platforms and mailing lists, namely Yahoo Answers, Stack Exchange Network, Quora, ResearchGate, ArchStudio, and Pallaido. These platforms were chosen for their substantial user bases and extensive repositories of software architecture-related content. The selection of keywords, including "Architecture Description Language(s)", "Software Architecture Tools", and "Software Architecture", was informed by an exhaustive literature review and a pilot study of 150 posts. Through iterative refinement, we identified these keywords as most relevant to the scope of our work.

Data collection was facilitated through a combination of API utilization and manual crawling techniques. While APIs provided streamlined access to platform data, manual crawling was employed in cases where APIs were unavailable or insufficient. The collected data, encompassing questions and answers, was stored in various formats, including CSV, HTML, and TXT. To standardize the data format for text analysis purposes, we developed PHP scripts to convert CSV and HTML files to TXT format, aligning with the requirements of the RQDA tool.

Quality assurance measures were implemented to ensure the integrity and reliability of the collected data. This involved leveraging insights from a comprehensive literature review to establish a database of software architecture-related problems highlighted in prior research. This database

Table 3. Data Acquisition from various sites / mailing lists

Sr. No.	Site / Mailing List	Questions
1	Stack Overflow	189
2	Programmers	35
3	Software Engineering	10
4	Yahoo Answers	27
5	Quora	68
6	ResearchGate	05
7	ArchStudio (mailing list)	1019
8	Palladio-dev (mailing list)	1091

served as a guiding framework throughout the coding and categorization process of posts, enhancing the accuracy and relevance of our analysis. Additionally, we capitalized on the features offered by community question answering sites to mitigate the presence of fake posts. Platforms such as Stack Overflow provided mechanisms for marking questions as duplicate and enabling users to vote on the credibility of questions and answers. In our research, we disregarded questions marked as deleted, duplicate, or receiving negative scores, thereby enhancing the credibility of our dataset. Our initial data set of Q&A contains a total of 2455 (given in Table 3) questions and answers. The data cleaning process is applied to delete the irrelevant Q&A. After cleaning, the final data set contains 282 questions.

3.2 Data Pre-Processing

We pre-processed data prior to analysis in R. R TextMining (TM) Library (Group, 2012) is utilized for pre-processing data and to perform content analysis. For data pre-processing we used R Text Mining (TM), Snowball packages and our customized functions. Data cleaning is a step of data pre-processing. Pre-processing makes data ready for further analysis. Text data is un-structured and contain noise like numbers, punctuation, email addresses, web links etc. Data cleaning is necessary to delete the unwanted text from the data. Data cleaning process prevent memory hogged. Data cleaning also involves removing punctuation's, URLs, email addresses, extra white-spaces (strip), special characters, stop words, HTML Tags and stemming words to root words. We performed stemming for text analytics section only. Stemming and Lemmatization are alternate options. Stemming is a crude process whereas, lemmatization is performed in a systematic way. The usage of stemming and lemmatization depends on particular use case. Where semantic information about words is required, lemmatization is a good choice as compared to stemming. In our case, our goal is to get an overview of collected data quickly and find the possible relationships between terms.

We used the text analytics to test the quality of collected data before performing in-depth manual analysis, i.e. grounded theory. Therefore, we opted for stemming by using Snowball Stemmer. We converted all documents to lower case and in plain text format. In addition to this, we also developed a small dictionary to remove words that are associated to Q&A sites vocabulary like (reputation, score, tag, answer-count, question-count, comment, votes, creation date, etc.). These words appear in each post. These words were selected from post of CQA sites manually and added into dictionary. Removal of these words from corpus improves quality of results as these words do not add any value and agitate knowledge extraction using text mining.

3.3 Data Analysis

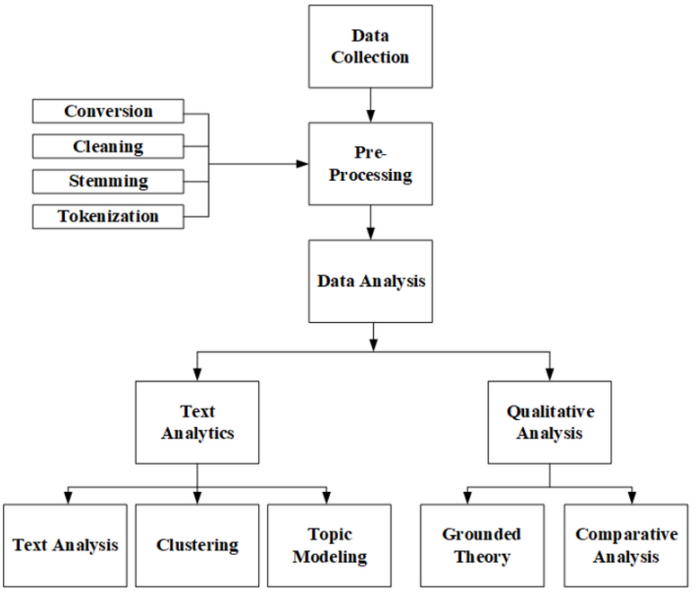
After data pre-processing we analyzed the data. We perform three types of analysis i.e. text analysis, topic modeling and qualitative analysis on collected data.

3.3.1 Text Analysis

We compiled a comprehensive corpus comprising all questions and answers (Q&A) and subjected it to content mining using the R TextMining Library. This process involved constructing a term-document matrix to represent the frequency of uni-gram, bi-gram, and tri-gram tokens within the corpus. Uni-grams are single words that appear in the text, while bi-grams are pairs of adjacent words, and tri-grams are sequences of three adjacent words. Analyzing these token sets helps in understanding the frequency, distribution, and context of words and phrases used in the dataset.

Subsequently, we conducted an array of analyses on these token sets, including term frequency charting, word cloud generation, and network analysis. Term frequency charting involves visualizing the frequency of words or phrases in the dataset, allowing us to identify the most common terms and their distribution. Word clouds visually represent the frequency of words in a dataset, with larger font sizes indicating higher frequencies. These visualizations served as valuable indicators of the informational richness of the selected Q&A dataset for our investigation.

Figure 2. Detailed Research Methodology



Moreover, hierarchical clustering techniques were employed to group similar tokens, facilitating the identification of common themes or topics within the dataset. Hierarchical clustering is a method for grouping similar data points into clusters based on their similarities. It helps in organizing and structuring the dataset, revealing underlying patterns and relationships among the data.

Additionally, to elucidate the interconnections between categories, we utilized a network analysis tool called Gephi. Network analysis involves studying the relationships between entities represented as nodes in a network. It helps in understanding the complex interactions and dependencies among different elements in the dataset, uncovering hidden structures and patterns. Overall, these analytical techniques provided valuable insights into the content and structure of our collected dataset, enabling us to validate its suitability for in-depth analysis.

3.3.2 Topic Modeling

Topic modeling, an unsupervised machine learning technique, holds significant value in extracting crucial insights from data without the need for manual review. Central to this technique is Latent Dirichlet Allocation (LDA), a widely recognized method within the realm of topic modeling. LDA has garnered substantial attention from researchers across diverse fields, as evidenced by its adoption in previous studies (Allamanis and Sutton, 2013; Arwan et al., 2015; Barua et al., 2014; Campbell et al., 2013; Bazelli et al., 2013; Bosu et al., 2013; de Souza et al., 2014) for mining Community Question Answering (CQA) sites. Leveraging LDA, we extracted essential topics from our corpus, illuminating key themes embedded within the dataset. However, it's important to note that LDA does not inherently assign labels to extracted topics; rather, it furnishes a list of words associated with each topic. To address this, we employed a pragmatic approach by assigning topics to words based on their frequency, with the top four terms serving as pivotal indicators. The Mallet implementation of LDA (McCallum, 2002) was utilized for topic extraction, with the determination of the optimal number of topics guided by coherence values.

Prior to running LDA, standard pre-processing steps, as outlined in the Data Pre-Processing subsection, were meticulously executed to ensure data quality and consistency. This included procedures such as text normalization, tokenization, and removal of stop words and irrelevant

characters. In the context of our research, topic modeling assumes a critical role in streamlining the coding process. By generating a comprehensive list of keywords and topics derived from the dataset, topic modeling empowers coders with valuable insights, thereby expediting the coding and categorization process. This proactive approach not only enhances efficiency but also augments the accuracy and reliability of the coding endeavor, ultimately saving valuable time and resources.

3.3.3 Qualitative Analysis

Qualitative analysis of the data was conducted utilizing Grounded Theory methodology, as proposed by Corbin and Strauss (Corbin and Strauss, 1990). This rigorous approach involved a systematic examination of the dataset, with authors meticulously scrutinizing each line of text to discern and assign codes to significant points. Codes were attributed to terms, phrases, and sentences deemed pertinent to the research inquiry, adhering to the principles of open and axial coding. This iterative process of coding, facilitated by the R-Qualitative Data Analysis (RQDA) tool (Chandra and Liang, 2016), enabled the systematic organization and categorization of data elements.

Grounded Theory methodology emphasizes the iterative nature of qualitative analysis, wherein codes emerge from the data itself rather than being predefined by the researcher. Open coding involves the initial exploration and identification of concepts within the data, while axial coding focuses on establishing relationships between these concepts. Through this process, codes are iteratively refined and grouped into categories, capturing the underlying structure and interconnections within the dataset.

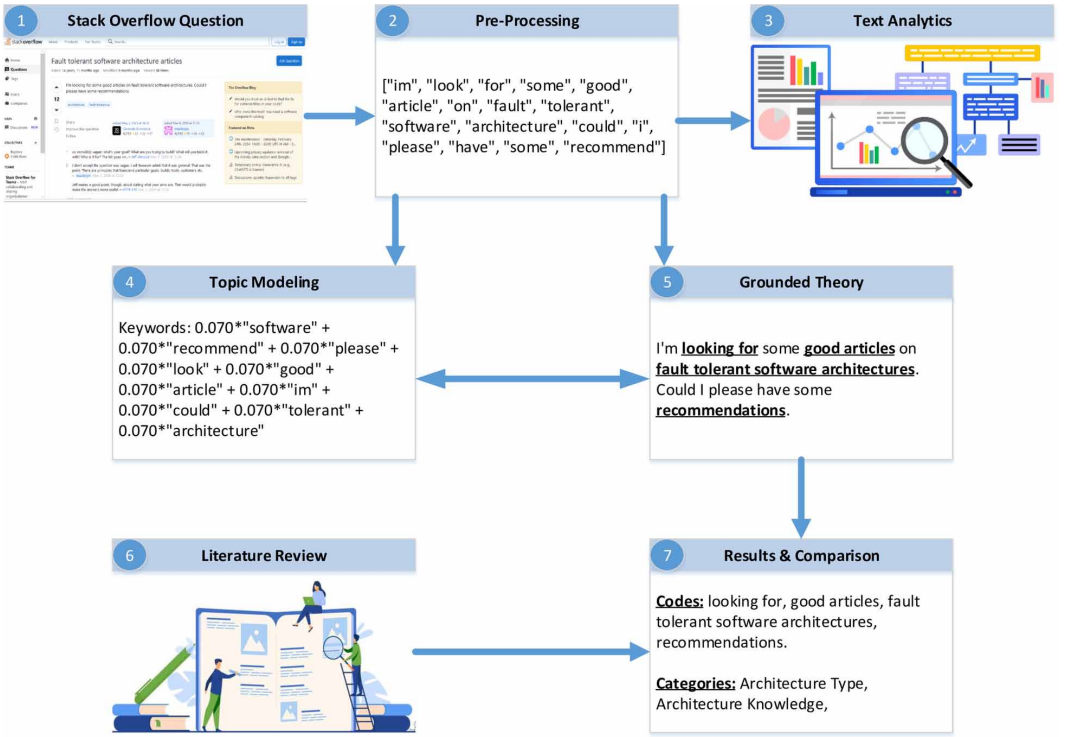
An essential aspect of Grounded Theory analysis is its capacity to generate rich conceptual frameworks from empirical data. The output of this methodology typically consists of a network of interconnected codes and categories, visually depicted through network plots. These plots serve as visual representations of the relationships and connections between different concepts, providing insights into the underlying structure of the dataset.

The RQDA tool facilitates the execution of various aspects of Grounded Theory analysis, including coding, categorization, and visualization of results. By leveraging the features offered by RQDA, researchers can effectively navigate the complexities of qualitative data analysis, ultimately yielding robust and meaningful insights into the phenomena under investigation.

4. RESULTS AND ANALYSIS

In this section, we begin by explaining the working of our proposed model through an illustrative example. Following this, we present the outcomes derived from each module within the methodology. The simulation of our research method- ology on a Stack Overflow question is visually depicted in Figure 3. Initially, a Stack Overflow question was collected in CSV format. Subsequently, a conversion to TXT format was undertaken for subsequent analysis. To facilitate this conversion, dedicated PHP scripts were developed, as TXT format aligns optimally with text analysis requirements and serves as the default format for the RQDA tool. Post-conversion, a data cleansing procedure was applied to eliminate extraneous Q&A instances. In the instance depicted, wherein a solitary question was selected, the cleansing step was deemed un- necessary. Pre-analysis data preparation was then undertaken in R, leveraging the capabilities of the R TextMining (TM) Library (Group, 2012). For data preprocessing, a combination of the R Text Mining (TM) and Snowball packages, along- side bespoke functions, were employed. This preprocessing phase rendered the data amenable for subsequent analysis. Notably, text data inherently carries structural irregularities and extraneous elements such as numerical values, punctuation, email addresses, web links, etc. Thus, data preprocessing assumed a pivotal role in purging the dataset of such superfluous elements. Furthermore, the preprocessing endeavor encompassed the removal of punctuation marks, URLs, email addresses, excess whitespace (strip), special characters, stop words, HTML tags, and the stemming of words to their root forms. Following the completion of the second step, the dataset is ready for further analysis.

Figure 3. An example of proposed framework simulation



In the third step, text analytics was employed to assess the integrity of the gathered data prior to embarking on comprehensive manual analysis, specifically grounded theory methodology. Additionally, a tailored dictionary was curated to filter out terms intrinsic to Q&A site vernacular, such as "reputation," "score," "tag," "answer-count," "question-count," "comment," "votes," "creation date," among others, which recurrently feature in each post. These terms, identified through manual curation of CQA site posts, were incorporated into the dictionary to facilitate their removal from the corpus. The elimination of such terms enhances result quality by eliminating extraneous information that does not contribute to knowledge extraction through text mining. Subsequently, content mining was executed on questions utilizing the R TextMining Library. Following the creation of the term-document matrix, uni-gram, bi-gram, and tri-gram tokens were extracted. Term frequency analysis, word cloud visualization, and network analysis were conducted on these token sets to assess the significance of the selected Q&A dataset for the study. Plots depicting word frequencies derived from uni-gram, bi-gram, and tri-gram tokens served as validation tools to ascertain the presence of requisite content within the collected dataset before proceeding with in-depth analysis. Hierarchical clustering techniques were deployed to categorize similar tokens, while network analysis utilizing the Gephi tool (Bastian et al., 2009) was employed to discern interconnections between categories.

In the fourth step, we employed Latent Dirichlet Allocation (LDA) to uncover salient topics within the question under analysis. Mallet (McCallum, 2002), an implementation of LDA, was employed to extract topics from the question. In the specific example under consideration, the outcomes of the LDA analysis are elucidated in step four of Figure 3. These LDA-derived topics were instrumental in aiding coders during the qualitative analysis phase utilizing grounded theory methodology. By providing a preliminary overview of the dataset, the results of the LDA analysis streamlined the decision-making process for the coders, thereby enhancing efficiency and efficacy in subsequent coding endeavors.

In the fifth phase, we applied grounded theory to discern key terms, referred to as codes, and subsequently allocated these codes to pertinent categories. To ensure the integrity and thoroughness of our data analysis, we adopted two complementary methodologies. Initially, we conducted a comprehensive literature review to identify prevalent issues pertaining to software architecture, compiling a database of challenges highlighted by prior research. This database served as a guiding framework throughout the coding and categorization process of the posts. Additionally, leveraging the outcomes of the LDA analysis described earlier, we expedited the coding process. In the provided example, codes and their associated categories are highlighted in boldface. The codes designated by coders in the aforementioned example include "looking for," "good articles," "fault-tolerant software architectures," and "recommendations." These codes were subsequently categorized into two overarching themes: "Architecture Type" and "Architecture Knowledge."

In the sixth step, we embark on an extensive review of published literature to identify and enumerate various concerns related to architecture. These concerns are meticulously scrutinized and cataloged, encompassing a wide array of topics ranging from design principles to scalability and beyond. Subsequently, in the seventh step, we meticulously compare these identified concerns with the results gleaned from the grounded theory approach. This comparative analysis serves as a crucial mechanism for validating the findings of our research in light of the existing body of literature. By comparing our findings with established literature, we aim to corroborate the robustness and validity of our research outcomes, ensuring that our contributions are firmly grounded in the broader academic literature.

In the example provided, the identified codes and their corresponding categories are subjected to rigorous validation against the framework proposed by Mayer et al. (Mayer and Weinreich, 2019) in the realm of architecture knowledge management. This validation process involves scrutinizing the alignment between our findings and the insights put forth by Mayer et al., thereby corroborating the relevance and applicability of our research outcomes within the existing scholarly landscape.

In the next sections, the results of text analytics, topic modeling (LDA) and grounded theory to analyze the information on the entire corpus are given. Text analytic comprises of analyzing the uni-gram, bi-gram and tri-gram tokens. We utilized term frequency chart, word cloud, network analysis and hierarchical clustering. The majority of the content examination is performed by composing an R language script. While, we utilized grounded theory which is a qualitative technique to perform information examination and extraction of useful information.

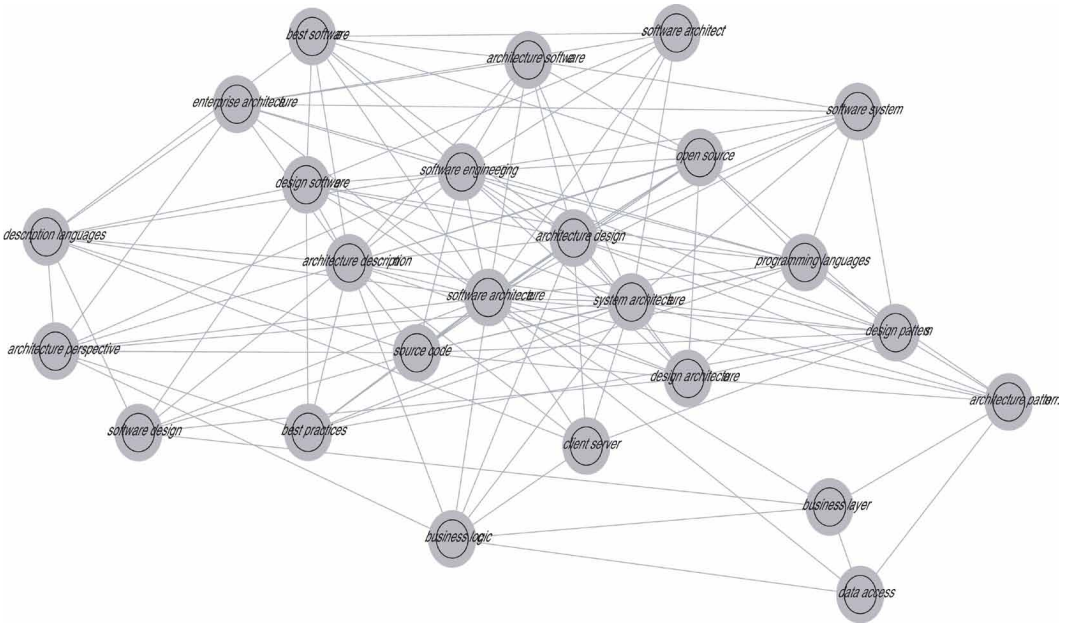
4.1 Text Analytic

We conducted an analysis of the corpus using various techniques including word frequencies, word clouds, and net- work plots utilizing both Term Frequency (TF) and Term Frequency Inverse Document Frequency (TF-IDF) methods. Our analysis revealed that 'Software Architecture,' 'Architecture Design,' and 'Architecture Description' were the most frequently occurring bi-gram tokens based on term frequency. Similarly, these terms were also prominent when utilizing TF-IDF. Additionally, 'Software Architecture Design,' 'Difference Software Architecture,' and 'Architecture Description Languages' emerged as the most commonly occurring tri-gram tokens in the corpus.

For the sake of brevity, we have uploaded the figures depicting uni-gram, bi-gram, and tri-gram tokens to our online project repository, which can be accessed via GitHub Repository[2]. These visual representations provide a clearer insight into the distribution and frequency of terms within the corpus.

Word cloud visualization offers an intuitive method to identify the most prominent terms within a text corpus. By visually representing word frequencies, word clouds assign weights to words based on their occurrence frequencies, thereby highlighting the key concepts and themes present in the dataset. In our analysis, terms such as 'Software,' 'Design,' 'Architecture,' 'WADL,' and 'System' emerged as the most frequently encountered in the word cloud of uni-gram tokens. Similarly, 'Programming Architecture,' 'Architecture Description,' and 'Architecture Design' were prominently featured as bi-gram tokens within the QA corpus.

Figure 4. Network Graph of Bigram Terms



The utilization of word clouds serves the purpose of gaining insights into the composition and thematic focus of the dataset. By examining the most prevalent terms, researchers can gain a preliminary understanding of the dominant topics and areas of interest encapsulated within the corpus. To facilitate further exploration and analysis, we have made the word cloud plots available online, allowing stakeholders to visually inspect and interpret the distribution of terms within the dataset. These visual representations offer a convenient means to assess the contents and thematic patterns of the dataset, thereby aiding in subsequent data-driven decision-making processes.

Social Network Analysis (SNA) stands as a versatile and invaluable tool for visualizing and dissecting textual corpora. Leveraging SNA, we constructed social network graphs representing both uni-gram and bi-gram terms derived from the corpus, employing both Term Frequency (TF) and Term Frequency Inverse Document Frequency (TF-IDF) approaches. These graphs offer a holistic view of the interrelationships among the 40 most frequent terms within the dataset.

In Figure 4, we present the social network graph depicting the connections between bi-gram tokens. Notably, the graph unveils intricate patterns of connectivity, shedding light on the prevalent themes and topics encapsulated within the corpus. Among the observed connections, prominent terms such as 'Software Engineering,' 'Software Architecture,' 'Architecture Design,' 'Architecture Description,' and 'System Architecture' emerge as highly interconnected nodes. This observation suggests a recurring theme in user queries, indicating a substantial interest and engagement with these topics within the community.

Conversely, terms like 'Data Access,' 'Business Layer,' and 'Business Logic' are depicted as sparsely connected nodes within the network graph. This finding suggests that these concepts appear less frequently in user inquiries, potentially indicating a narrower scope or less prevalent interest among users. By scrutinizing the social network graphs, researchers can glean valuable insights into the underlying structure and dynamics of the corpus. These visualizations serve as powerful tools for identifying key themes, detecting patterns of interaction, and uncovering latent relationships among terms. As such, they offer a robust framework for conducting in-depth analyses and informing strategic decision-making processes. To unravel the intricate relationships among uni-gram, bi-gram,

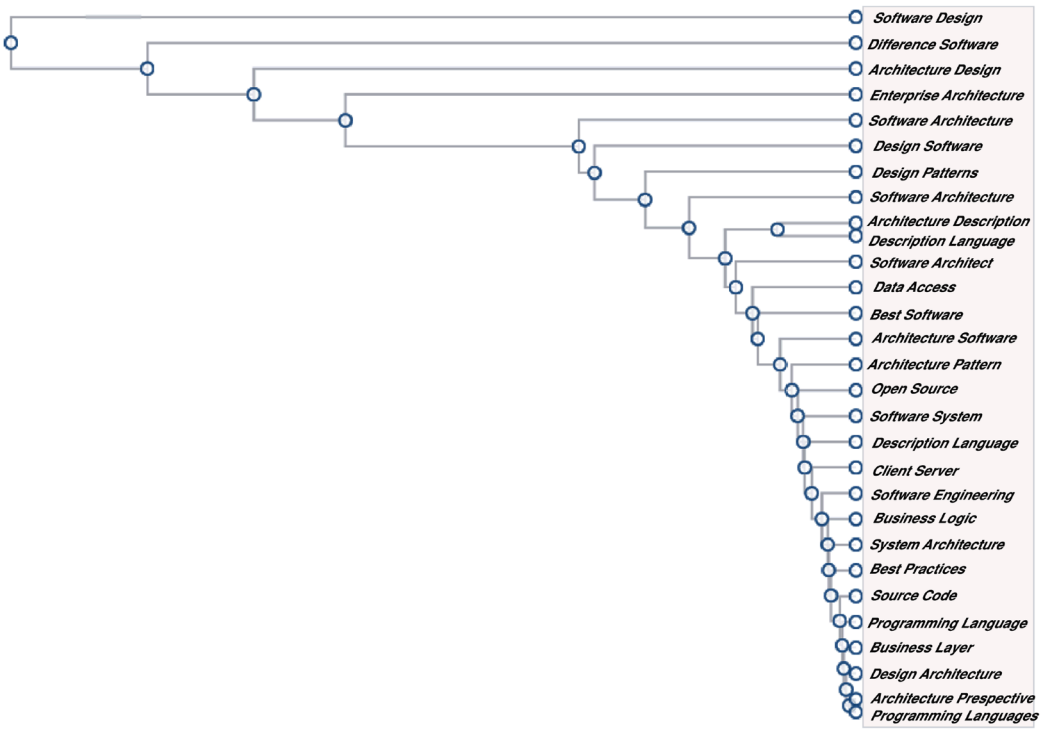
and tri-gram tokens within our dataset and to categorize them into cohesive clusters, we employed hierarchical clustering methodology. This analytical approach allowed us to identify tokens sharing similar semantic contexts, facilitating the discernment of recurring themes and prevalent topics within the corpus.

In our investigation, the hierarchical clustering of bi-gram tokens revealed notable patterns, with 'Software Design' emerging as a central theme intricately linked with various other terms within the cluster. Additionally, terms like 'Different Software,' 'Architecture Design,' 'Enterprise Architecture,' 'Design Software,' and 'Design Patterns' formed distinct clusters, indicative of prevalent topics discussed within the community's Q&A interactions. Furthermore, the grouping of terms such as 'Software Architecture,' 'Software Design,' 'Architecture Description,' and 'Design Patterns' within the same cluster suggested a strong semantic association, implying their frequent co-occurrence and potential significance in discussions on optimal design solutions. Visual representation of this clustering process can be found in Figure 5, offering insights into the corpus's underlying structure and prevalent discourse patterns.

4.2 Grounded Theory

By utilizing RQDA Tool we applied Grounded Theory on the Q&A. At first, we read every post and utilized open coding to code all the critical content of Q&A. At that point we combined the related code and aggregate 94 codes were recognized by the authors. Next, we analyzed the codes and made 14 code categories and relegated related code to these categories. The coding process was finished by the three authors. Every author involved in coding also checked the coding of other author and in case of disagreement codes were re-assigned. The fourth author supervised & reviewed the coding and settle the 14 coding conflicts by combining the comparable codes and expelling the

Figure 5. Hierarchical clustering of bi-gram terms



duplicate ones. Top codes that were observed by the authors are given in Table 4. Codes are given in first column of table, second column of table is occurrence frequency of code and in the third column number of post is given in which code appears. The most of the question i.e. (70+41=111) that practitioners asked are about the clarification of architecture concepts. It means there are many confusions about the concepts of architecture among the community. The third most popular code is about architecture styles. 36 codes are about the understanding of architecture. In these questions users shared their architecture problem and want further clarification of their shared architecture. 36 codes are related to architecture patterns. There are 30 occurrences of architecture tools related questions. 29 questions are about the design decisions like selection of suitable architecture style or pattern for a particular problem.

The visualization of grounded theory, depicted in Figure 6, provides a comprehensive overview of the categories and codes derived from our analysis. Within this visual representation, categories are denoted by colored circles, while individual codes are depicted in white. Notably, categories such as Advantages of Architecture, Architecture Knowledge, Architecture Description Languages, and Designing Architecture emerge as prominent focal points, reflecting the primary areas of questions among practitioners. To delve deeper into the intricate relationships between these categories and codes, we leveraged Gephi, a powerful network visualization tool. By exporting the data from RQDA to Gephi, we generated a Network Graph showcasing the interconnectedness of codes and categories, as illustrated in Figure 6.

Upon examination of the Network Graph, a distinct pattern emerges: a significant portion of practitioner inquiries revolves around Architecture Tools, reflecting a collective interest in enhancing proficiency and skill-sets within this domain. Moreover, the visualization highlights the interconnected nature of certain categories, such as Software Architecture Patterns, Knowledge of Architecture, and Advantages of Architecture, indicating overlapping themes and shared concerns within the architectural discourse. Notably, these categories are linked through common codes, underscoring the multifaceted nature of architectural challenges and the nuanced interplay between various concepts.

Furthermore, our analysis reveals that a substantial number of issues—61 out of 94—span multiple categories, under- scoring the complexity and interconnectedness of architectural concerns. Specifically, 28 issues are associated with more than three categories, with an additional 15 issues addressing more than three categories concurrently. For a comprehensive list of these multifaceted

Figure 6. Grounded Theory Network

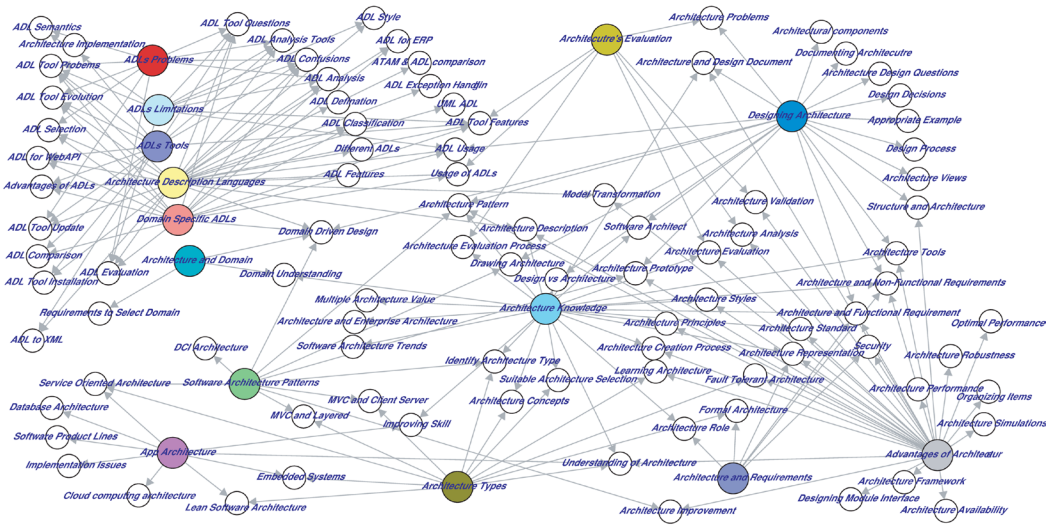


Table 4. Top Codes

Code	Frequency	Files
Learning Architecture	70	66
Architecture Questions	41	36
Architecture Styles	40	37
Understanding of Architecture	36	30
Architecture Patterns	36	32
Architecture Tools	30	22
Design Decisions	29	25
Software Architecture	27	27
Suitable Architecture Selection	27	25
Architecture Design	17	11
ADL Tool	16	13
Drawing Architecture	16	16
Identify Architecture Type	15	15
Architecture Evaluation	13	10
MVC and Layered	13	11
Software Architect	13	9
Architecture Problems	12	12
Documenting Architecture	12	10
Implementation Issues	12	12

issues and their respective category affiliations, please refer to the supplementary materials provided at 3. Within Figure 7, we have highlighted issues that pertain to more than three categories, offering insights into the breadth and depth of architectural challenges faced by practitioners.

The concept map illustrates various interconnected concepts related to software architecture. At the center of the map is the overarching concept of the "Advantages of Architecture," which serves as a focal point for understanding the benefits associated with architectural practices in software development. Branching out from this central node are several key concepts, including "Architecture and Requirements," emphasizing the relationship between architectural decisions and project requirements, and "Understanding of Architecture," highlighting the importance of comprehension in architectural design. These nodes are complemented by "Architecture Types," which denotes the different architectural styles or paradigms that developers may employ.

Furthermore, the concept map delves into the practical aspects of architectural design, with nodes such as "App Architecture" and "Designing Architecture" focusing on the application and implementation of architectural principles. The map also encompasses aspects related to architectural description and documentation, as indicated by nodes like "Architecture Description" and "Architecture Description Languages," underscoring the significance of clear and concise architectural documentation. Additionally, the map recognizes specialized areas within architectural practice, such as "Architecture Patterns" and "Domain Specific ADLs," reflecting the diversity and specialization within the field. Overall, the concept map provides a comprehensive overview of the interconnected concepts that constitute the domain of software architecture, highlighting their interdependencies and significance in software development processes.

4.3 Topic Modeling

We used Latent Dirichlet Allocation (LDA) for topic modeling. LDA being a statistical topic modeling technique can be used as an alternative of Grounded Theory. The difference in LDA and Grounded theory is that LDA is fully automated and automatically extract topic keywords. Whereas, Grounded Theory is a manual techniques which require reading of whole corpus. The topic keywords extracted by LDA can be compared with codes of Grounded Theory whereas, categories of Grounded Theory can be compared with LDA topics. LDA results can also verify the quality of analysis performed by using Grounded Theory. Mallet (McCallum, 2002) which is implementation of LDA is used to extract topics from the corpus. We ran the LDA for 10, 20 and 30 topics. Maximum Coherence Score 0.466 is achieved by running LDA for 30 topics its means 30 topics are in the corpus. List of all topics is given in Table 5. Topics are numbered from 0 to 29. Topic number 1, 2, 5 and 26 are related to post attributes of CQA sites. These topics can be deleted because they add no value to the analysis. From this we can say that there are 26 significant topics in the corpus. Layered Architecture with Topic Rate of 0.542 is most prominent topic on the selected CQA sites. Features of Architecture Tool with Topic Rate of 0.539 is second most prominent topic. A list of top eight topics is given in Table 6. From this list we can observe that most of statistically significant architecture problems are related with the architecture patterns, tools features, documentation and definition.

5. EVALUATION AND DISCUSSION

We further compared our results with the findings of various research papers (Malavolta et al., 2013; Fuxman, 2000; Garlan et al., 2010; Woods and Bashroush, 2012; Mishra and Dutt, 2005; Bradbury et al., 2004; Kamal and Avgeriou, 2007; Khan et al., 2016; Capilla et al., 2016; Shahin et al., 2014; Clements, 1996; Keeling, 2015; Medvidovic and Taylor, 2000; Ozkaya and Kloukinas, 2013; Woods and Hilliard, 2005; Patwardhan and Patwardhan, 2016; Schriek et al., 2016; Taylor, 2019; Deryugina et al., 2019; Cai and Kazman, 2019; Mayer and Weinreich, 2019; Wan et al., 2023). This section compares the findings of various researchers and pertinent post that we have identified from CQA sites. This segment additionally demonstrates that our findings are in accordance with the existing research. But our method of research is unique as compared with the existing research on the topic and unlike previous research which is restricted to one topic/ problem, we covered the broad range of issues. The comparison is given in Table 7. First column of the table is issues that are recognized from the literature, Second column is depiction of issue explained in the literature. The third column is reference to pertinent code that we distinguished through grounded theory during coding from community posts lastly, analysis of the issue is also given. The analysis depends on the perspectives of the community users, i.e. we return to the pertinent posts and based on the feedback of users propose the solution of the problem. The analysis describes the reason for the issue and it will be helpful for future researchers. Each column of the table is an independent research area that will add to the knowledge of software architecture.

The primary findings of this research are as seventeen recognized issues and their proposed solution. We grouped these issues into five broader concepts. A brief description of each concept and proposed solution is summarized below:

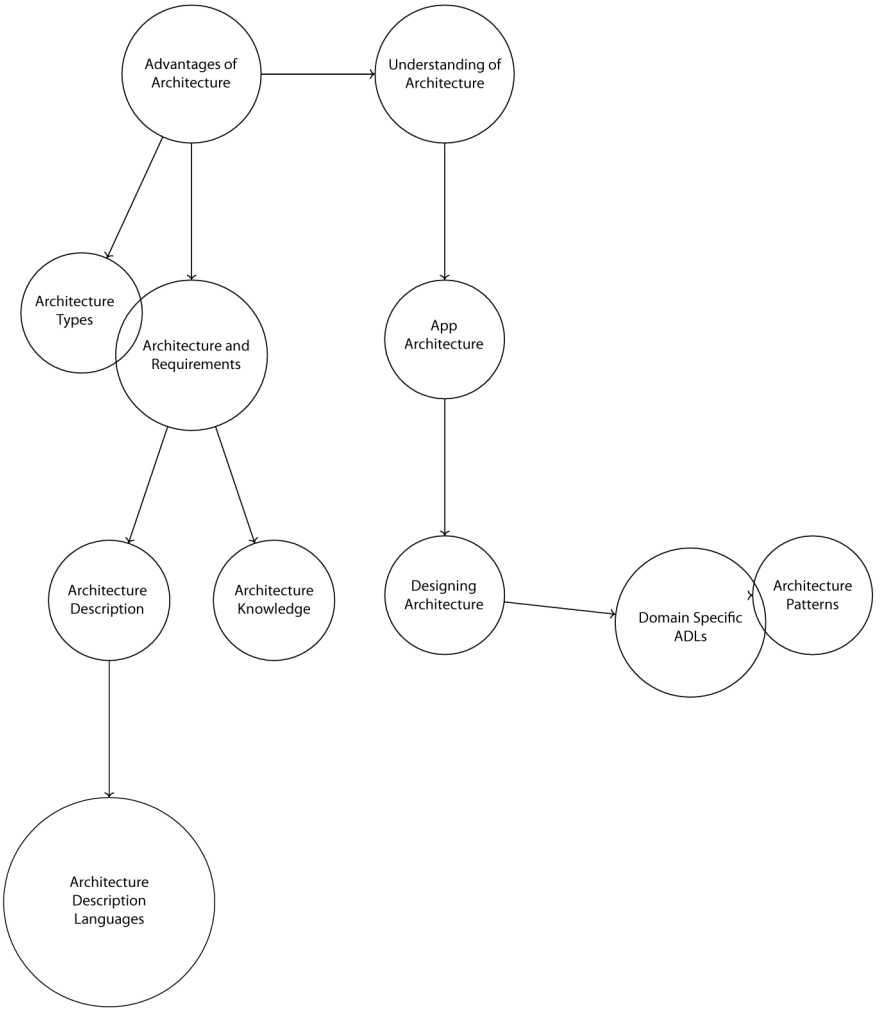
- **Knowledge Management and Standardization:** The first category encompasses efforts aimed at managing and standardizing architectural knowledge. IT professionals often encounter difficulties in accessing relevant information and standardized representations of architectural concepts. Thus, the study advocates for the creation of databases and knowledge management systems to capture known issues and solutions. Additionally, standardizing architecture representations and notations can streamline communication and comprehension among practitioners, facilitating more efficient collaboration and problem-solving.

Table 5. Topics extracted from Corpus

ID Topic	Words	Rate
0 Database Application Design	database, application, design, good, follow, web, multiple, company, destination, pattern	0.411
1 Question Attributes	answercount, tag, work, closeddate, score, body, software, deletiondate, commentcount, lasteditdate	0.476
2 Post Attributes	architecture, body, commentcount, favoritecount, creationdate, deletiondate, answercount, parentid, lasteditdate, viewcount	0.416
3 Architecture Literature	good, system, model, resource, book, service, question, product, top, approach	0.290
4 Technology Time	time, year, technology, system, provide, work, product, student, current, approach	0.188
5 Answer Attributes	viewcount, body, parentid, posttypeid, tag, owneruserid, creationdate, commentcount, score, favoritecount	0.666
6 Diagram and Code Request	answer, diagram, code, vote, uml, architect, work, detail, case, mar	0.302
7 Generate Code	code, view, create, controller, method, add, problem, object, event, require	0.289
8 Software Architecture Tool	tool, software, architecture, source, run, good, verticle, present, support, call	0.305
9 Component Interface	interface, component, myx, archstudio, framework, brick, architecture, make, connector, object	0.238
10 Archstudio Questions	file, interface, component, brick, archstudio, create, class, project, architecture, select	0.309
11 Web Services	web, rest, api, soap, service, db, context, standard, net, protocol	0.281
12 Software Code	code, software, project, deletiondate, base, implement, public, body, score, closeddate	0.331
13 System Architecture	class, make, system, architecture, work, person, file, problem, implement, project	0.206
14 Architecture Documentation	architecture, document, score, owneruserid, favoritecount, title, viewcount, closeddate, commentcount, describe	0.529
15 Layered Architecture	layer, datum, object, business, model, entity, access, tier, domain, public	0.542
16 Service Oriented Architecture	service, architecture, feature, description, project, yesod, archstudio, site, process, file	0.308
17 Architecture Software	architecture, software, system, difference, enterprise, term, question, style, model, soa	0.410
18 Client Server Architecture	server, client, web, application, app, framework, side, user, develop, template	0.421
19 Software Application	software, view, application, operation, design, question, architect, pattern, follow, stuff	0.344
20 Data Types	type, int, size, byte, return, mem, function, memory, create, variable	0.365
21 Architecture Information	architectural, architecture, system, information, make, project, book, people, thing, interested	0.278
22 Architecture Solution	datum, user, architecture, solution, separate, point, token, project, approach, update	0.420
23 Module Design	module, application, design, message, architecture, define, type, project, gt, object	0.317
24 Features of Architecture Tool	architecture, software, system, requirement, design, find, structure, part, build, documentation	0.539
25 Learning Architecture Software	design, learn, architecture, software, good, write, pattern, answer, application, language	0.362
26 Post Attributes	title, creationdate, parentid, favoritecount, posttypeid, lasteditdate, answer-count, body, viewcount, closeddate	0.613
27 Architecture Definition	component, create, type, structure, interface, outer, connector, signature, archipelago, open	0.510
28 Architecture Description Languages	architecture, software, language, question, description, adl, computer, compare, topic, tool	0.328
29 Object Oriented	class, public, connection, endpoint, abstract, method, inherit, ftgw, client, source	0.313

- **Integration and Flexibility:** Integration challenges and the need for architectural flexibility constitute another significant category of concerns. Software integration issues can hinder system interoperability and performance, necessitating robust modeling languages and tools to address these complexities. Moreover, designing flexible architectures capable of accommodating evolving requirements and changes in software development processes is crucial for long-term viability and maintainability.

Figure 7. Concept Map of issues belonging to more than three categories



- **Gamification and Skill Enhancement:** Recognizing the importance of continuous skill enhancement in software architecture, this category emphasizes the role of gamification and innovative learning approaches. Gamification mechanisms offer engaging platforms for IT professionals to enhance their design skills and deepen their under- standing of architectural concepts. By incorporating gamified elements into educational initiatives, practitioners can be incentivized to actively participate in skill-building activities, thereby fostering a culture of continuous learning and improvement.
- **Tool Support and Collaboration:** The category of tool support and collaboration underscores the significance of leveraging technology and fostering collaborative efforts within the software architecture community. Developing tools equipped with architecture knowledge sharing capabilities can enhance information dissemination and problem-solving efficiency. Additionally, bridging the gap between researchers and practitioners through collaborative initiatives and standardization efforts is essential for advancing the field and addressing common challenges effectively.

Table 6. Top Eight Topics

ID	Topic	Rate
15	Layered Architecture	0.542
24	Features of Architecture Tool	0.539
14	Architecture Documentation	0.529
27	Architecture Definition	0.510
18	Client Server Architecture	0.421
22	Architecture Solution	0.420
0	Database Application Design	0.411
17	Architecture Software	0.410

- **Visualization and Representation:** Visualization techniques and standardized representations play a pivotal role in conveying complex architectural concepts and facilitating informed decision-making processes. This category high- lights the importance of exploring visualization methods for architecture evaluation and enhancing comprehension among stakeholders. Furthermore, advocating for standardized architectural representations using languages such as ADLs and UML can promote consistency and clarity in architectural documentation, enabling more effective communication and collaboration.

5.1 Evaluation With LLMs

Evaluation of the results of the proposed model was also performed with Generative Pre-trained Transformer (GPT) and Bidirectional Representation of Architectures in Transformers (BRAD). The query "please give me the concerns or problems that IT professionals face during designing software architecture" was submitted to both GPT and BRAD to retrieve responses related to the concerns and problems encountered during the design of software architecture. GPT and BRAD were tasked with generating responses to the query based on their understanding of the domain and the data they were trained on. The retrieved responses were then compared to identify similarities and differences in their representations of the concerns and problems faced by IT professionals in designing software architecture.

The comparison results presented in Table 8 highlight the responses generated by the Proposed Model, GPT, and BRAD to the query regarding concerns and problems faced by IT professionals during software architecture design. The table provides insights into the areas of agreement and divergence between the models, shedding light on their understanding of the challenges inherent in this domain.

5.2 Areas of Agreement Between the Proposed and LLMs

Knowledge Management: All three models acknowledge the importance of knowledge management in software architecture design, indicating a shared recognition of the need to effectively manage and disseminate architectural knowledge within organizations.

Integration Issues: The models agree on the existence of integration issues during software architecture design, reflecting a common understanding of the challenges associated with integrating diverse components and systems to create cohesive software solutions.

Integration of Non-Functional Requirements (NFRs): The models recognize the significance of integrating non- functional requirements, such as performance, security, and scalability, into software architecture design, underscoring the importance of addressing these factors to ensure the success of software projects.

Table 7. Comparison and Analysis

Issue	Literature	Categorization and Analysis
Architecture standardization	Informal representation of architecture components and behavior (). Modeling concerns ad- dressed by standardization ().	Standardization of representation required for clarity and formality in architecture diagrams. Queries about architecture representation, drawing, and learning can be ad- dressed through standardization.
Project delay	Limited focus on architecture leads to software project delays (Malavolta et al., 2013).	Categorization of questions needed for solution database maintenance to avoid adhoc development.
Software Integration	Incompatible XML schemas pose integration challenges ().	Development of software solutions for architecture integration required. UML Refactoring tool () may assist.
Non Functional Requirements (NFR)	Handling NFRs at project end compromises quality (Khan et al., 2016).	Integration of NFRs at architecture level discussed for learning purposes.
Flexible Architecture	Design flexibility for accommodating changes essential (Keeling, 2015).	Tools like UML Refactoring (Deryugina et al., 2019) and DV8 (Cai and Kazman, 2019) can achieve flexibility.
Design learning	Card games and gamification tested for teaching design (Schriek et al., 2016; Mayer and Weinreich, 2019).	Games and gamification mechanisms useful for learning and understanding architecture.
Visualization Techniques (VTs)	Limited evaluation of VTs for software architecture ().	Knowledge from CQA sites can evaluate architecture and ADL tools.
ADLs developed in Isolation Architecture Knowledge Management	ADLs developed in isolation lack business context (Taylor, 2019; Garlan et al., 2002). Architecture Knowledge Management challenges (Mayer and Weinreich, 2019; Wan et al., 2023; Capilla et al., 2016).	ADLs need comparison and classification based on do- main for selection and evaluation.
Suitable ADL selection	Difficulties in comparing ADLs due to diverse features (Medvidovic and Taylor, 2000).	Classification and comparison of ADLs required based on domain and features for selection.
Limited use of ADL	ADLs not mainstream due to formal analysis, usability, and reliability challenges (Ozkaya and Kloukinas, 2013).	Queries on UML ADL and usage of ADLs for substitution to UML. ADLs need to be feature-rich and dependable.
Academic perspective	ADLs developed primarily for academic use (Fuxman, 2000).	Interest in ADLs for architectural implementation, learning, and understanding. Need for industry-oriented ADLs.
Gap between practitioners and research community	Communication gap between practitioners and researchers limits ADL usage (Bradbury et al., 2004).	Queries on ADL usage for ERP systems and API writing. Need for industry-oriented ADLs.
ADL standardization	Lack of general-purpose ADLs due to lack of standardization (Woods and Bashroush, 2012).	Standardization necessary for general-purpose and domain-specific ADLs.
Enterprise data analysis	Challenges in analyzing large enterprise data volumes (Woods, 2005).	Queries on architecture description and tools for enterprise application development and analysis.
UML ADL	UML elements found in recent ADLs, but deficiencies remain (Clements, 1996; Wan et al., 2023).	Queries on architectural representations using ADL & UML. ADLs need to address issues with business model- ing.
Standard for system behavior	Lack of universal standards for system behavior in ADLs ().	Standardization needed to classify ADLs into groups for better understanding.

Table 8. Comparison of Proposed Model with GPT and BARD

Knowledge Management	Yes	Yes	Yes
Standard Notations	Yes	Yes	-
Integration Issues	Yes	Yes	Yes
Integration of NFRs	Yes	Yes	Yes
Flexible Architecture	Yes	Yes	Yes
Gamification	Yes	-	-
Tool Support	Yes	-	Yes
Visualization	Yes	-	-
ADLs for Academia	Yes	-	-
ADLs Selection	Yes	-	-
Limited usage of ADLs	Yes	-	-
Professional Tools	Yes	-	-
Research and Practical GAP	Yes	-	Yes
ADLs Standardization	Yes	-	-
Enterprise Application Develop-	Yes	Yes	-
Ment			
Architecture Representation	Yes	Yes	Yes
Architecture Levels	Yes	-	Yes

5.3 Areas of Divergence Between the Proposed and LLMs

Gamification: While the Proposed Model highlights the potential benefits of incorporating gamification techniques into software architecture design, responses from GPT and BRAD on this concern are not available, suggesting a potential gap in their understanding of this aspect.

Visualization: Only the Proposed Model emphasizes the importance of visualization techniques in software architecture design. Responses from GPT and BRAD on this concern are not available, indicating a potential oversight in their representation of this aspect of software architecture.

The comparison results also reveal several areas where responses from GPT and BRAD are not available, indicating potential limitations or gaps in their understanding of certain concerns and problems related to software architecture design. This underscores the need for further investigation and refinement of these models to ensure their comprehensive coverage of the domain.

Overall, the comparison provides valuable insights into the strengths and limitations of the Proposed Model, GPT, and BRAD in representing the concerns and problems faced by IT professionals during software architecture design. By analyzing the areas of agreement and divergence between the models, we can gain a deeper understanding of their respective capabilities and areas for improvement, informing future research and development efforts in this field.

6. CONCLUSION AND FUTURE WORK

In this research, we reviewed different papers about software architecture. The principal motivation behind our examination was to recognize issues/problems related to software architecture. At first, we gathered different issues of software architecture from research papers. Next, to distinguish the issues looked at by software engineers, we gathered information from different CQA sites and mailing lists.

We utilized different text analytics to feature the key terms and utilized grounded theory to feature and categorize the issues. We compared our outcomes and the results of different research papers for benchmarking. We additionally proposed solutions to the distinguished issues gathered from the research papers and the collection of Q&A data.

In addition to the issues identified in Section 5, during the coding of posts, we observed that most people are confused about software architecture, and some are mixing the basic concepts of software architecture. Most of the questions asked are related to literature requests, suitable examples, and knowledge of architecture. The community is confused about the usage of various architectural patterns and styles. Various people also ask about selecting suitable architecture for a specific problem. Being frequent tri-gram tokens, ADLs are not a popular topic on Q&A sites, and there are very limited questions about ADLs. The most frequently asked questions about ADLs are about the advantages of ADLs, domain- driven design, ADL tools, limitations of ADLs, evaluation of ADLs, and domain-specific ADLs.

In the future, we plan to extend our analysis of Q&A sites by analyzing the behavior of the software architect's community. We also plan to extend our analysis by applying topic modeling to research papers and community posts. After that, we will try to fill the gap in published literature on various software architecture topics and work to find solutions to these topics posted on community question-answering sites by software engineers.

ACKNOWLEDGMENT

We thank our colleagues who motivated and helped us complete this research. Special thanks to all the anonymous reviewers working day and night to facilitate the researchers by providing valuable feedback to improve the quality of the research.

DISCLOSURE STATEMENT

The authors declare that they have no conflict of interest.

FUNDING STATEMENT

No funding was received for this work.

PROCESS DATES

Received: January 4, 2024, Revision: March 15, 2024, Accepted: March 13, 2024

CORRESPONDING AUTHOR

Correspondence should be addressed to Zeeshan Anwar (zeeshan0333@yahoo.com)

REFERENCES

Allamanis, M., & Sutton, C. (2013). Why, when, and what: analyzing stack overflow questions by topic, type, and code, *Proceedings of the 10th Working Conference on Mining Software Repositories*, IEEE Press, pp. 53–56. doi:10.1109/MSR.2013.6624004

Anwar, Z., & Afzal, H. (2024). Mining crowd sourcing repositories for open innovation in software engineering. *Automated Software Engineering*, *31*(1), 11. doi:10.1007/s10515-023-00410-z

Anwar, Z., Afzal, H., Ahsan, A., Iltaf, N., & Maqbool, A. (2023). A novel hybrid cnn-lstm approach for assessing stackoverflow post quality. *Journal of Intelligent Systems*, *32*(1), 20230057. doi:10.1515/jisys-2023-0057

Arwan, A., Rochimah, S., & Akbar, R. J. (2015). Source code retrieval on stackoverflow using lda, *Information and Communication Technology (ICoICT), 2015 3rd International Conference on*, IEEE, pp. 295–299.

Bajaj, K., Pattabiraman, K., & Mesbah, A. (2014). Mining questions asked by web developers, *Proceedings of the 11th Working Conference on Mining Software Repositories*, ACM, pp. 112–121. doi:10.1145/2597073.2597083

Barua, A., Thomas, S. W., & Hassan, A. E. (2014). What are developers talking about? an analysis of topics and trends in stack overflow. *Empirical Software Engineering*, *19*(3), 619–654. doi:10.1007/s10664-012-9231-y

Bass, L., Clements, P., & Kazman, R. (2003). *Software architecture in practice*. Addison-Wesley Professional.

Bastian, M., Heymann, S., & Jacomy, M. (2009). Gephi: An open source software for exploring and manipulating networks. *Proceedings of the ... International AAAI Conference on Weblogs and Social Media. International AAAI Conference on Weblogs and Social Media*, *8*(1), 361–362. doi:10.1609/icwsm.v3i1.13937

Bazelli, B., Hindle, A., & Stroulia, E. (2013). On the personality traits of stackoverflow users, *Software maintenance (ICSM), 2013 29th IEEE international conference on*, IEEE, pp. 460–463.

Bosu, A., Corley, C. S., Heaton, D., Chatterji, D., Carver, J. C., & Kraft, N. A. (2013). Building reputation in stackover- flow: an empirical investigation, *Mining Software Repositories (MSR), 2013 10th IEEE Working Conference on*, IEEE, pp. 89–92.

Bradbury, J. S., Cordy, J. R., Dingel, J., & Wermelinger, M. (2004). A survey of self-management in dynamic software architecture specifications, *Proceedings of the 1st ACM SIGSOFT workshop on Self-managed systems*, ACM, pp. 28– 33. doi:10.1145/1075405.1075411

Cai, Y., & Kazman, R. (2019). Dv8: automated architecture analysis tool suites, *Proceedings of the Second International Conference on Technical Debt*, IEEE Press, pp. 53–54.

Campbell, J. C., Zhang, C., Xu, Z., Hindle, A., & Miller, J. (2013). Deficient documentation detection: a methodology to locate deficient project documentation using topic analysis, *Proceedings of the 10th Working Conference on Mining Software Repositories*, IEEE Press, pp. 57–60. doi:10.1109/MSR.2013.6624005

Capilla, R., Jansen, A., Tang, A., Avgeriou, P., & Babar, M. A. (2016). 10 years of software architecture knowledge management: Practice and future. *Journal of Systems and Software*, *116*, 191–205. doi:10.1016/j.jss.2015.08.054

Chandra, Y., & Liang, E. S. (2016). *Qualitative Data Analysis with RQDA*. Springer Singapore.

Clements, P. C. (1996). A survey of architecture description languages, *Proceedings of the 8th international workshop on software specification and design*, IEEE Computer Society, p. 16. doi:10.1109/IWSSD.1996.501143

Corbin, J. M., & Strauss, A. (1990). Grounded theory research: Procedures, canons, and evaluative criteria. *Qualitative Sociology*, *13*(1), 3–21. doi:10.1007/BF00988593

de Souza, L. B., Campos, E. C., & Maia, M. A. (2014). Ranking crowd knowledge to assist software development, *Proceedings of the 22nd International Conference on Program Comprehension*, ACM, pp. 72–82. doi:10.1145/2597008.2597146

Deryugina, O., Nikulchev, E., Ryadchikov, I., Sechenev, S., & Shmalko, E. (2019). Analysis of the anywalker software architecture using the uml refactoring tool. *Procedia Computer Science*, *150*, 743–750. doi:10.1016/j.procs.2019.02.005

Di Pompeo, D., & Tucci, M. (2023). Quality attributes optimization of software architecture: Research challenges and directions, *2023 IEEE 20th International Conference on Software Architecture Companion (ICSA-C)*, IEEE, pp. 252– 255.

Dimov, A., Emanuilov, S., Bontchev, B., Dankov, Y., & Papapostolu, T. (2022). Architectural approaches to overcome challenges in the development of data-intensive systems. *Human Factors in Software and Systems Engineering*, *61*, 38. doi:10.54941/ahfe1002521

Fuxman, A. D. (2000). *A survey of architecture description languages, Technical report*. Penn State.

Galster, M., & Weyns, D. (2023). Empirical research in software architecture—Perceptions of the community. *Journal of Systems and Software*, *202*, 111684. doi:10.1016/j.jss.2023.111684

Garlan, D., Cheng, S.-W., & Kompanek, A. J. (2002). Reconciling the needs of architectural description with object- modeling notations. *Science of Computer Programming*, *44*(1), 23–49. doi:10.1016/S0167-6423(02)00031-X

Garlan, D., Monroe, R., & Wile, D. (2010). *Acme: An architecture description interchange language, CASCON First Decade High Impact Papers*. IBM Press. doi:10.1145/1925805.1925814

Group, R. I. (2012). *Mining Your Qualitative Text, Technical report*. R Interest Group.

Guamán, D., Pérez, J., Diaz, J., & Cuesta, C. E. (2020). Towards a reference process for software architecture recon- struction. *IET Software*, *14*(6), 592–606. doi:10.1049/iet-sen.2019.0246

Hasselbring, W. (2018). *Software architecture: Past, present, future, The Essence of Software Engineering*. Springer.

Ho-Quang, T., Chaudron, M. R., Hebig, R., & Robles, G. (2020). *Challenges and directions for a community infrastruc- ture for big data-driven research in software architecture*. Model Management and Analytics for Large Scale Systems.

Kamal, A. W., & Avgeriou, P. (2007). An evaluation of adls on modeling patterns for software architecture, *Proceed- ings of the 4th International Workshop on Rapid Integration of Software Engineering Techniques (RISE 2007). LNCS. Springer, Heidelberg*.

Keeling, M. (2015). Lightweight and flexible: Emerging trends in software architecture from the saturn conferences. *IEEE Software*, *32*(3), 7–11. doi:10.1109/MS.2015.65

Khan, F., Jan, S. R., Tahir, M., Khan, S., & Ullah, F. (2016). Survey: Dealing non-functional requirements at architecture level. *VFAST Transactions on Software Engineering*, *9*(2), 7–13. doi:10.21015/vtse.v9i2.410

Land, R. (2002). *A brief survey of software architecture, Technical report*. Malardalen University.

Malavolta, I., Lago, P., Muccini, H., Pelliccione, P., & Tang, A. (2013). What industry needs from architectural languages: A survey. *IEEE Transactions on Software Engineering*, *39*(6), 869–891. doi:10.1109/TSE.2012.74

Mayer, B., & Weinreich, R. (2019). The effect of gamification on software architecture knowledge management: a student experiment and focus group study, *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, ACM, pp. 1731–1740. doi:10.1145/3297280.3297449

McCallum, A. K. (2002). Mallet: A machine learning for language toolkit, http://mallet. cs. *umass. edu*.

Mckenzie, F. D., Petty, M. D., & Xu, Q. (2004). Usefulness of software architecture description languages for modeling and analysis of federates and federation architectures. *Simulation*, *80*(11), 559–576. doi:10.1177/0037549704050185

Medvidovic, N., & Taylor, R. N. (2000). A classification and comparison framework for software architecture description languages. *IEEE Transactions on Software Engineering*, *26*(1), 70–93. doi:10.1109/32.825767

Mishra, P., & Dutt, N. (2005). Architecture description languages for programmable embedded systems. *IEE Proceedings. Computers and Digital Techniques*, *152*(3), 285–297. doi:10.1049/ip-cdt:20045071

Navasa, A., Pérez-Toledano, M. A., & Murillo, J. M. (2009). An adl dealing with aspects at software architecture stage. *Information and Software Technology*, *51*(2), 306–324. doi:10.1016/j.infsof.2008.03.009

OMG (2007). OMG Unified Modeling Language Specification.

Othmane, L. B., & Lamm, M. (2019). Mindset for software architecture students, *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*, Vol. 2, IEEE, pp. 306–311.

Ozkaya, M., & Kloukinas, C. (2013). Are we there yet? analyzing architecture description languages for formal analysis, usability, and realizability, *2013 39th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*, IEEE, pp. 177–184.

Rehman, N., & Khan, A. W. (2022). *Critical Challenges of Designing Software Architecture for Internet of Things (IoT) Software System*. John Wiley Sons, Ltd., URL https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119821779.ch11 doi:10.1002/9781119821779.ch11

Schriek, C., van der Werf, J. M. E., Tang, A., & Bex, F. (2016). Software architecture design reasoning: A card game to help novice designers, *European Conference on Software Architecture*, Springer, pp. 22–38. doi:10.1007/978-3-319-48992-6_2

Seifermann, S., Ta¸spolatoˇ, T., Reussner, R. and Heinrich, R. (2018). Challenges in secure software evolution-the role of software architecture.

Shahin, M., Liang, P., & Babar, M. A. (2014). A systematic review of software architecture visualization techniques. *Journal of Systems and Software*, *94*, 161–185. doi:10.1016/j.jss.2014.03.071

Soliman, M., Galster, M., & Avgeriou, P. (2021). An exploratory study on architectural knowledge in issue tracking systems, *European Conference on Software Architecture*, Springer, pp. 117–133. doi:10.1007/978-3-030-86044-8_8

Tamburri, D., Kazman, R., & Van Den Heuvel, W.-J. (2019). Splicing community and software architecture smells in agile teams: An industrial study, *Proceedings of the 52nd Hawaii International Conference on System Sciences*. doi:10.24251/HICSS.2019.843

Taylor, J. T., Taylor, W. T., Taylor, J. T., & Taylor, W. T. (2021). Software architecture, Patterns in the Machine: A Software Engineering Guide to Embedded Development pp. 63–82.

Taylor, R. N. (2019). *Software architecture and design, Handbook of Software Engineering*. Springer.

Tian, F., Liang, P., & Babar, M. A. (2019). How developers discuss architecture smells? an exploratory study on stack overflow, *2019 IEEE International Conference on Software Architecture (ICSA)*, pp. 91–100. doi:10.1109/ICSA.2019.00018

Villegas, N., Tamura, G., & Müller, H. (2017). *Architecting software systems for runtime self-adaptation: Concepts, models, and challenges, Managing trade-offs in adaptable software architectures*. Elsevier. doi:10.1016/B978-0-12-802855-1.00002-2

Wan, Z., Zhang, Y., Xia, X., Jiang, Y., & Lo, D. (2023). Software architecture in practice: Challenges and opportunities, *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 1457–1469. doi:10.1145/3611643.3616367

Whiting, E., & Andrews, S. (2020). Drift and erosion in software architecture: summary and prevention strategies, *Proceedings of the 2020 the 4th International Conference on Information System and Data Mining*, pp. 132–138. doi:10.1145/3404663.3404665

Woods, E. (2005). Architecture description languages and information systems architects: Never the twain shall meet? *Artechra white paper*.

Woods, E., & Bashroush, R. (2012). Using an architecture description language to model a large-scale information system–an industrial experience report, *Software Architecture (WICSA) and European Conference on Software Archi- tecture (ECSA), 2012 Joint Working IEEE/IFIP Conference on*, IEEE, pp. 239–243.

Woods, E., & Hilliard, R. (2005). *Architecture description languages in practice session report, WICSA 5*. IEEE.

Zhao, Y. (2016). Architecture: Description Really Matters. *Skyscrapr*, *4*, 1–6.

ENDNOTES

1 https://github.com/zeeshan0333/Architecture
2 https://github.com/zeeshan0333/Architecture
3 https://github.com/zeeshan0333/Architecture