Selecting Software Architectural Styles from Requirement Documents Using NLP

by

Javeria Yasmin

NS 00000329575

Supervisor

Asst. Prof Dr. Yawar Abbas Bangash

A thesis submitted to the faculty of Computer Software Engineering Department, Military College of Signals, National University of Sciences and Technology, Rawalpindi in partial fulfilment of the requirements for the degree of MS in Software Engineering

February 2023

Thesis Acceptance Certificate

This is to certify that the final copy of the thesis written by NS **Javeria Yasmin**, Registration no: **00000329575,** Course: **MSSE-27,** on the topic, **"Selecting Software Architecture Styles from Requirement Documents using NLP"** has been found complete. The plagiarism report is satisfactory and necessary amendments pointed out by GC members are incorporated.

Signature: _____

Name of Supervisor: Asst Prof Dr. Yawar Abbas Bangash

Date: _____

Signature (HOD): _____

Date: _____

Signature (Dean/Principal): _____

Date: _____

Declaration

I, Javeria Yasmin declare that this thesis titled **"Selecting Software Architecture Styles from Requirement Documents Using NLP"** and the work presented in it is my own and has been generated by me as a result of my own original research.

I confirm that:

1. This work was done wholly or mainly while in candidature for a Master of Software Engineering degree in MCS NUST.

2. Where any part of this thesis has previously been submitted for a degree or any other qualification at NUST or any other qualification at NUST or any other institution this has been clearly stated.

3. Where I have consulted the published work of others, this is always clearly attributed.

4. Where I have quoted from the published work of others the source is always given. With the exception of such quotations this thesis is entirely my own work.

5. I have acknowledged all main sources of help.

6. Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

_____

Javeria Yasmin.
00000329575
MSSE27

Dedication

“In the name of Allah, the most Beneficent, the most Merciful"

I dedicate this thesis to my parents, teachers, friends, and teachers who supported me each step of the way.

Acknowledgments

All praise to Allah for the strength and blessings to complete my thesis.

I would like to convey my gratitude to my supervisor, Asist. Prof. Dr. Yawar Abbas Bangash, for his supervision and constant support. His continuous help, constructive comments and suggestions throughout the experimental and thesis works are major contributions to the success of this research. Also, I would thank my committee members; Assoc. Prof. Dr. Hammad Afzal, and Lt. Col Khawir Mehmood for their support and knowledge regarding this topic.

Last, but not the least, I am highly thankful to my parents. They have always stood by my side and have been a great source of inspiration for me. I would like to thank all of them for their support through my times of stress and excitement.

Abstract

With the growth in the size and complexity of softwares, the design phase of software systems has recently drawn more attention. Software architecture is a fundamental idea in this phase and has a significant impact on the software development life cycle, with the degree to which it is utilized frequently determining the success of a software project. Software architecture prediction is the crucial step before the implementation phase. To solve the inherent issues with current methods that have been reported in the literature, this research makes a novel method based on quality attributes to assess software architecture design. The PURE dataset has been used to extract the quality parameters from Software Requirement Specification (SRS) document. Natural Language Processing (NLP) and Machine Learning (ML) techniques have been used to bring automation in architecture styles prediction with efficiency and accuracy. The Automated Architecture Style Prediction (AASP) system has minimized the architect role, and 93.75 accuracy has been achieved through Artificial Neural Network Algorithm. Architecture Style designing can be automated in future and can consider more architecture styles.

Table of Contents

Page

List of Tables

List of Figures

Chapter 1

Introduction

1.1. Overview

After the requirement specification is complete and shortly before the designing phase, the planning of the software architecture for any problem begins. The architecture phase is frequently referred to as the "EARLY DESIGN STEP" for this reason. A high level of abstraction must be used to depict a software system when planning and designing its architecture. Software architecture is a phase of the software development life cycle that specifies how to address a customer's issue in a way that satisfies both their functional and non-functional requirements.

Without specifying any implementation specifics, architecture should contain all managerial and operational elements. It marks the beginning of the solution domain. The problem-solving approach, or architecture, should also consider external product features like "efficiency," "portability," "usability," "reliability," "maintainability," "security," etc. [1] [2].

The architect was also expected to document the architecture after it was completed. IEEE ISO/IEC/IEEE 42010:2011 has replaced IEEE 1471 as the industry standard for software architecture. Unified Modelling Language (UML) notations, box and line diagrams, or other formal architectural description languages are the bases for several Architecture Description Languages (ADLs) that are available to define the architecture. ADLs include things like Architecture Description Mark-up Language, Aspect Oriented Architecture Description Language, Rapide, Aesop, and Architecture Analysis and Design Language, among others. [2].

The Architecture of Software is predicted following the essential guidelines below:

- An architecture of software system should simply satisfy the needs of the customers. It shouldn't provide any execution specifics.
- The software system's architecture should be created with the idea that it should be adaptable to incorporate the maintenance and any changes in requirements in future[3].

- The architecture must be created in a way that ensures the eventual product built upon it will meet all of the specifications and quality needs of the customer [3].
- The software architecture should make an effort to foresee every potential risk that can be posed to the system in the future [4].
- The architecture of a software system should support optimal system resource use.

The advantages of developing software architecture are described below.

- The initial stage of development, which outlines the problem-solving strategy, is software architecture. Because it demonstrates how to construct the system, it can provide information on the system's behaviour before it is actually put into use.
- Software architecture aids in resource estimation for the solution domain [5].
- The risk-prone places in the system can be quickly identified with the use of architecture. and can so lessen the risk's effect and expense. Risk management, in other words, happens during the design phase.
- Architecture of the software presents the components and modules of the system, making it simple to reuse components created for one system in another if the components are the same in both systems. Reusability is thus present as well [6].
- One may simply increase the system's quality with the aid of architecture, allowing one to create a product that possesses all desirable qualities, such as "maintenance," "reliability," "efficiency," "usability," etc. The system's structure as shown by the software architecture paper with communicating interfaces is depicted [7].

1.2. Motivation and Problem Statement

The system analysts read whole Software Requirement Specification documents and then after analysis specify a unique architectural Style for that specific project. Architectural style is defined based on the Non-Functional Requirements. The Architecture document of the software depicts the working of the system with interactive interfaces [7]. It also defines the quality related parameters like “maintainability”, “security”, “efficiency”, “reliability”,” usability” etc. of the system. The fundamental issue in architecture style prediction is reading huge requirements documents and then selecting a specific architecture. It’s a time consuming and hectic task. spectrum sensing is to precisely differentiate between a primary user and secondary user. There is a possibility for malice SU to mimic the signature of PU and dodge the system into believing that it’s a licensed primary user.

Hence, there is a need for an automated architecture style prediction mechanism. Once the document is fed into the system it will read the document and according to the requirements predict a suitable architecture style. It will also rank the top three most suitable styles for the document.

1.3. Objectives

The thesis is aimed to achieve the following objectives: -

- To propose an **Automated Architecture Style Predictor (AASP)** from the requirement documents.
- Identify challenges in identifying exact Non-Functional requirement.
- Identify different NLP and ML techniques to ensure the correct Non-Functional Requirements extraction and enables the extraction of ambiguous quality attributes or recommends the attributes based on the similar domain projects. It also trains the system with different domain projects through the application of ML
- **Transforming Architectural style prediction to Be Automated**: Requirements are extracted from the Software Requirement Specification (SRS) Document using NLP based strategies and feature extraction algorithms. Different ML based models will then be used on that structured data to predict the exact Architectural style, while considering all the quality attributes and other parameters associated.
- Bring Automation by using ML Techniques.
- Accurate Software Architectural Style prediction with no to minimal involvement of System Architect.

1.4. Thesis Contribution

The mechanism proposed in this research has not been used for predicting the architecture styles. Moreover, NLP and Machine Learning on software requirement documents is also applied in dataset generation and architecture style prediction which is also not applied in the existing work.

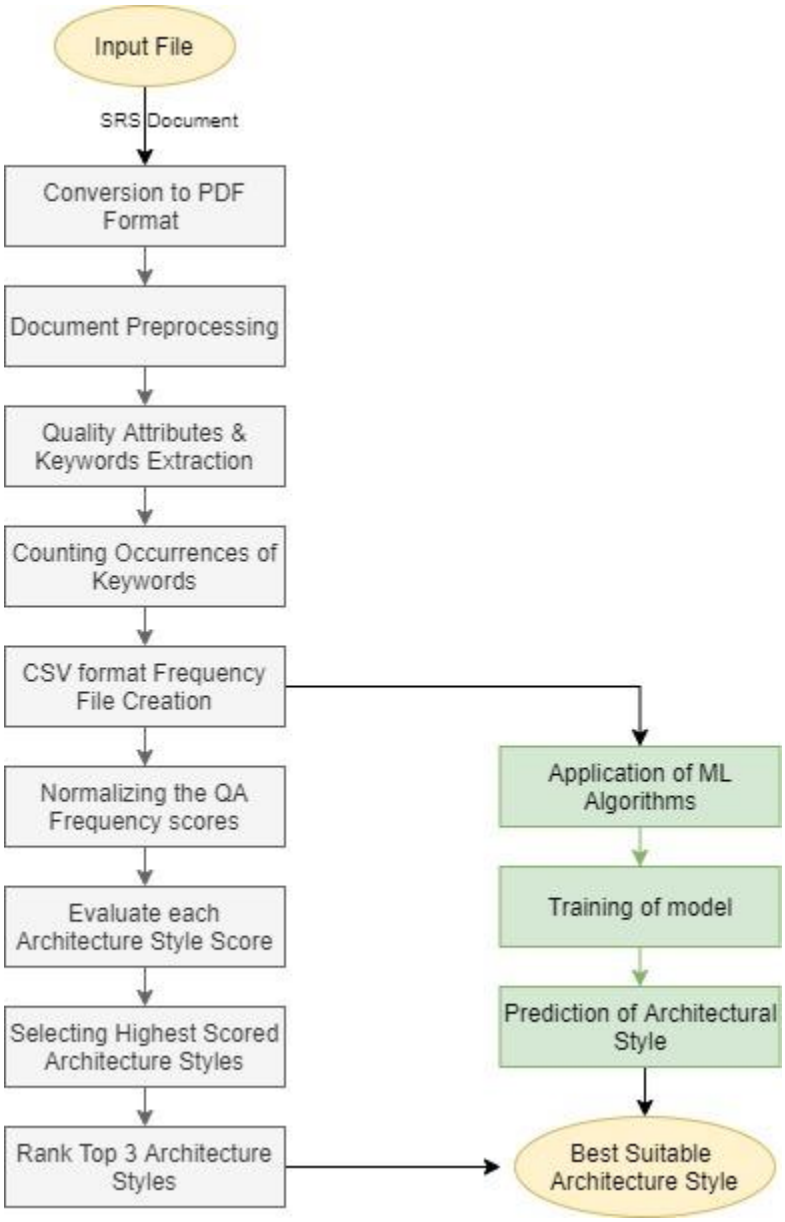

Figure 1. Block Diagram depicting the working of Automatic Architecture Style Predictor (AASP)

Once the document is inserted in the system, It will do some preprocessing and extract the desired Quality Attributes; then evaluate the architecture styles based on calculated scores and then rank the top three styles; Machine Learning models will run on the dataset created and with accuracy architecture styles are predicted.

This research has contributed the previous work as follow:

- Automated Architecture Style Predictor from a Software Requirement Specification document in mean time with accuracy has been proposed.
- Next, we have proposed a ranking mechanism in which the most suitable top three Architecture Styles are predicted against the project.

- Unlike existing work, we have not considered domain specific architecture styles, we keep it generic.

1.5. Thesis Organization

The structure of the thesis is as follow:

- Chapter 2 contains the literature reviewed in the thesis. The previous related work regarding NLP and Architecture Style Prediction is explained. Prediction and Evaluation techniques and existing case study-based approaches applied on the architecture styles are covered in the chapter.

- Chapter 3 covers the details of Architecture Styles and Quality Attributes under consideration in the AASP system.

- Chapter 4 contains the explanation of Natural Language Processing techniques and frameworks used.

- Chapter 5 contains the proposed scheme for AASP algorithm, details of Architecture Styles under consideration, quality keywords used for prediction system, formulae for architecture style score calculation. The simulation results representing the working of the scheme are covered in chapter 6.

- Chapter 6 contains the Machine Learning models executed on the dataset. The simulation results for each model and the attributes used are discussed in detail. The discussion and Analysis on the results is also the part of this chapter

- Chapter 7 is the concluding chapter. The conclusion and future research gaps are described in this chapter.

Chapter 2

Literature Review

There is a lot of research that has been done in Machine Learning and Natural Language Processing in the area of software engineering to automate the designing phase of Software Development Life Cycle and to assist developers. Automation work is mostly done in auto generation of UML diagrams through the extraction of keywords from the Functional Requirements and designing accordingly. Activity Diagram, Sequence Diagram, Usecase Diagram and Class Diagrams are automatically, and semi automatically generated using Natural Language Specifications with 100% accurate outcomes.[8] There is also some work in the Bi-directional Traceability feature between requirements of the system and design phase but need more advancement to completely automize using ML techniques.[9]

The NFRs are extracted through the document using NLP and ML techniques. NLP is used to extract Non-Functional Requirements related words and calculate the frequency of occurrence of the NFR in the document based on their word count. Domain vocabulary is constructed through which the NLP based phrase matching techniques were used to extract exact NFRs. Machine learning models are used for the training of the knowledge area through Support Vector Machine (SVM). SVM also maps the NFRs and Architectural concerns. Finally, it deduces the architectural utility concern spaces. This methodology gives 77% accuracy as compared to the expert system architect prediction of the quality attributes and architectural aspects.[10] Quality attributes are the important constituents of any software which are usually not given much importance but had great impact. For Agile based software development, a framework is proposed to extract the quality factors and aspects from user stories where requirements are more volatile. For the implementation of QA Extractor, NLP and regular expression techniques has been used. [17]

A multi criterion decision method has been proposed in which based on the quality attributes, it aids the different candidates to structure of architectures for a software system. Quality attributes were given weights based on their impact level on the system and sometimes through discussion with the stakeholders.[11] Weighted sum Approach was also proposed to bring accuracy. In that approach .com

lar software system according to different stakeholders' preference level. And then these preference scores were cumulatively calculated, to select the architecture.[12] Different Authors have worked on performance, modifiability, availability, safety, and security quality attributes for the selection of architectural style.[13]

Dataset for the software systems was also designed to be used for research related activities. Clustering algorithms were applied on that dataset for data analysis tasks.[14] Some work has been done in a semi-automated architectural style prediction in which NLP and applications of ML like ontology and neural network are used to identify the exact design against software requirements.[16] So, there is huge research gap in architectural style prediction and designing with minimal human involvement.

Different evaluation techniques and frameworks exist, each of which can be customized for certain objectives. The two types of current evaluation techniques are quantitative and qualitative. Research normally gives us some guidance by which we could infer qualitative features to employ software architecture [18]. A comparative technique to analyze the degree of fulfilling various quality characteristics in various field areas is through quantitative description. We can also say, quantitative estimation shows how significant the reported abilities and benefits are. The blend of each and every architectural style's innate specific quality attributes which makes up the qualitative evaluation of architectural styles [18, 19]. The techniques may be combined. The following provides more information on both qualitative and quantitative methodologies.
The most popular approaches for evaluating architecture regarding each quality in qualitative evaluation are questioning techniques. Measuring techniques are less comprehensive than questioning techniques, but they can be used to address specific problems and software features (such performance or scalability). Scenarios, questionnaires, and checklists are the three different categories of questioning approaches. Although there are significant differences between them in terms of their applicability, they all aim to improve our knowledge of the level of alignment between architecture and necessary quality features. The case-study based technique gives context to quality qualities (such performance, security, maintainability, and reliability). Scenarios are analytical techniques for assessing the value of characteristics in each

context. The generalized selection and somewhat open questions that are relevant to all the architectures are given in the questionnaire approach [20]. Some questions focus on the development and documentation of architecture, while others are more descriptive in nature. The checklist technique makes use of a series of extremely specific questions that were gleaned from multiple experiences on several related subjects. These inquiries concentrate on unique qualities that a system possesses. A checklist can help you maintain a balanced focus on all system components [21].

Measuring methods produce quantitative outcomes in quantitative evaluation. Instead of giving us answers to design-related concerns, these strategies address the measurement team's queries about the qualities of architecture. Compared to questioning tactics, these techniques are fuller and more sophisticated. Metrics are quantitative interpretations based on certain observable design metrics, such as component fan-in/fan-out [22]. The evaluation procedure for measuring techniques should pay attention to both the assumptions underlying their use as well as the results of the metric [20]. Building a prototype or simulating a system can assist in developing and explaining architecture, but they are frequently expensive. In other words, they frequently contribute to the process of development.

Software architecture is assessed using a variety of various techniques. To better comprehend architecture and demonstrate that it deals with functional needs in addition to qualitative ones, case study-based approaches like Scenario-based Architecture Analysis Method [23, 25] proposed in 1993. This approach was created in order to understand the risks connected with the architecture and conflicting possibilities between various quality attributes, as well as to make sure that the architectural notions are compatible with the system's desirable features. Qualitative qualities and the balance between the attributes are covered by other techniques, such as ATAM (Architecture Tradeof Analysis Method) [24, 25]. These techniques assess the architecture to demonstrate the degree to which particular quality objectives are accomplished. Additionally, they emphasize the contrast between qualitative characteristics and how they affect one another. On SAAM, ATAM is based.

Other expense-based approaches exist, such as Cost-Benefit Analysis Method [25, 26], which, in contrast to the more two approaches, has built a link between software

economic and development concerns. Expense, especially financial plan, is viewed as a qualitative element in this methodology. This method measures the relationship between monetary concerns and other qualitative architectural ideas. To examine the flexibility of the system, ALMA (Architecture-Level Analysis Method) [27, 28] employs threat assessment and maintenance cost estimation. A technique to gauge an information system family's expandability and interoperability is called FAAM (Family-Architecture Analysis Method) [29].

Formal techniques are the focus of another type of approaches. Different kinds of architectural language explanations, which are emblematic languages used to represent and describe the software architecture systems, are included in formal techniques [30]. Aesop [31], C2SADL [32], Darwin [33], MetaH [34], Rapide [35], UniCon [36], and ACME [37] are only a few of the different architectural languages that have been developed as a result of recent work on formal architectural language descriptions.

We will now explore the research on multi-criteria challenges for choosing an architectural style. An all-purpose instrument for decision-making in multiple criteria is the integral. A Decision Support System (DSS) is introduced by Moaven et al. [38, 39] that uses a fuzzy notion to convey concepts of quality attributes more precisely and effectively. The system makes use of fuzzy inference to help software architects' choices. To provide an environment for analyzing a software system's heterogeneous architecture styles, Moaven et al. [40] introduce a structure and a method. To strengthen the evaluation validity, the framework makes use of the quantity attributes that were acquired during their use. According to Babu et al. [41], the ANP technique is utilized to more precisely and effectively describe the numerous criteria and ideas of quality attributes. The optimization of software architecture design is one of the objectives of this strategy. As a strategy for the methodical and verifiable selection of architectural styles, Galster et al. [42] created SYSAS. This methodology is focused on the traits of fundamental architectural components that are important to developers as well as parts of the proposed system that final users can see. Making the most of the features of styles is the study's principal goal. In Zaki et al. [43], an integrated SPL approach with component-based design and architecture style selection is given. This method uses a mathematical analytic order technique to choose the optimum styles of architecture. A library of styles is

provided by Dwivedi and Rath [44] for choosing the best style for software systems. The formal modelling language alloy is used in this study to model reused and extensible complex and widely dispersed components. Tahmasebipour and Babamir [45] have assessed how architectural strategies and architectural styles interact. The greatest architectural style for each individual quality feature or their groupings can be found by utilizing a new ranking system for architectural styles. Tan and Chen [46] present an intuition-based fuzzy Choquet integral for multiple scenario-based choice making. These integrals account for interactions between the selection criteria. Babu et al. [47] made the recommendation for an enhanced selection approach known as SSAS, which reflects inter-dependencies among the assessment criteria using the analytical process within a zero-one objective standard of programming. For fuzzy-based numerical architectural evaluation, Dhaya and Zayaraz [48] introduce the ADUAK meta-model. The research is aimed to support architectural knowledge management (AKM) to increase the effectiveness of the architectural design process. Many choice issues, according to Saaty [49], cannot be organized taxonomically because they include interactions between and dependencies between top-level parts and lower-level elements. The challenges of independence on substitutes or benchmarks and dependence among substitutes or benchmarks are each solved using the AHP and ANP, respectively [50].

The most informative research when assessing the evaluation methodologies are those by Abraho and Insfran [51]. In this study, the well-known architecture tradeoff analysis method (ATAM) is compared to the quality-pushed architecture root and improvement (QUADAI) method, demonstrating that QUADAI achieves superior scores than ATAM in the stance of efficacy, observed usefulness, and objective to use.

Mahdavi-Hezavehi et al. [52] conducted some research to show which style of architecture manages qualitative traits to assist the investigators in understanding how to evaluate and give weightage to parameters in self-compliant systems. Other findings that might apply to architecture have been done in the context of qualitative characteristics of product lines of softwares [53]. To locate and analyze all research in the era of 1996 - 2010 that present quality features and metrics for SPL, this paper conducts a comprehensive literature analysis. The SDLC phase in which the measurements are implemented, the accompanying quality features, and their assistance for particular SPL

traits have all been taken into consideration when classifying these attributes and measures. Other strategies are offered by Moaven et al. [54] and Dasanayake et al. [55] and are helpful in making decisions regarding picking architectural styles. Moaven et al[38, 39] emphasis is on the decision-making process and employing information that may be helpful in this choice.

There are significant issues with all the methods listed, such as the overlapping or conflicting nature of the quality criteria. This is particularly important when choosing an appropriate architectural design for the modernization of a heritage system. Due to the complexity of selecting styles and the uncertainty involved, particular emphasis must be devoted to incorporating system architecture expertise and using learnable procedures. Binary thinking and numerical conceptions are insufficient to capture the true characteristics of a style. These kinds of issues can never be fully resolved, but there are always ways to increase the precision and potency of selecting the appropriate style. In this paper, we offer a solution to the problems currently facing society.

Table 1. NLP Techniques Literature Review

Author/s	**NLP Techniques**		
	Worked On	***Technique/s***	***Results***
Pang and Lee [15][16]	• Declare which sentences in the text are objective or subjective. • Prevents polarity categorization from taking into account any false information	• Machine learning classifier • Extraction method based on minimum-cut formulation	Inter-sentence level integration of information with a substantial amount of words
Maalej and Hadeer [17]	• Create categories for user reviews, issues, new features, user experiences, and ratings • Tense used, the rating, the	• NLP techniques • Sentiment analysis	Accuracy achieved 97%. of text of each review were

Author/s	NLP Techniques		
	Worked On	***Technique/s***	***Results***
	sentiment score, and the length		also considered.
Jacob and Harrison [18]	• Create a model that addresses feature requests in English-language, reviews the mobile applications. • Labeled feature data manually.	NLP techniques (Linguistic rules and classification)	
Yang and Liang [19]	• Reviewer comments on mobile apps can be used to identify the functional and nonfunctional requirements.	•Evaluation metrics: recall, precision, and F-measure •Combination of TF-IDF and NLP techniques.	
Guzman and Maalej [20]	• Reviewer comments are used to extrapolate software product features. • Helpful to product managers for setting the highest priority for upcoming releases. • discover faulty software product components	NLP techniques and sentiment analysis	
Mujahid et al. [21]	• Study on the reviews of a mobile application that was carried out to investigate and classify consumer complaints.	NLP Probabilistic approaches	
Chaochang Chiu [22]	• Chinese-language reviews were extracted from mobile game applications and analyzed.	Statistical Approaches	Analyze variables such as game

Author/s	NLP Techniques		
	Worked On	***Technique/s***	***Results***
			characteristics, gender, and game type.
Kilani et al. [23]	• Classification of reviews of applications for the healthcare field. • Problems, new characteristics, sentiment analysis, general bugs, usability, security, and performance are the issues being addressed.	Utilizing several machine learning and NLP techniques with the Weka tool	

Chapter 3

Architectural Styles

3.1 Architectural Styles

Architectural styles and designs outline how to arrange system components in order to develop a complete system and fulfill the customers' requirements. Knowing which architectural style and pattern would be most suitable for your project is crucial given the variety of architectural patterns and styles available in the software industry. Table 2 provides a comprehensive classification of all current

Table 2: Architecture Styles Categorization

Application Type	Architectural Style
Shared Memory	1. Blackboard 2. Data-centric 3. Rule-based
Distributed System	1. Client-server 2. Space based architecture 3. Peer-to-peer Distributed System 4. Shared nothing architecture 5. Broker 6. Representational state transfer 7. Service-oriented
Messaging	1. Event-driven Messaging 2. Asynchronous messaging 3. .Publish-subscribe
Structure	1. Component-based 2. Pipes and filters 3. Monolithic application based 4. Layered
Adaptable System	1. Plug-ins 2. Reflection 3. Microkernel

Modern System	1. Architecture for Grid Computing 2. Multi-tenancy Architecture 3. Architecture for Big-Data

3.1.1. Blackboard style

The primary elements in the blackboard-style are depicted in Figure 2. A Blackboard shared repository in here depicted. Several knowledge sources use it to address the issue. Each expert attempts to address the issue and records their full or partial resolution on the whiteboard. The solution provided by the preceding knowledge source is simultaneously modified or expanded by all other knowledge sources, or each knowledge source presents the solution in its own unique way [56]. As a result, several knowledge sources collaborate to find a solution. Control shells are used to plan and direct the actions of the knowledge sources so that no hiccups may be made that would cause the project to go off course from its intended course. All of these operations are managed, controlled, and coordinated by the control shell during a problem-solving session. Scalability, or the capacity to add or remove sources of knowledge from the system as needed, is one of the benefits of this architecture. Because sources of knowledge sources are not reliant on each other, they can operate simultaneously with a governing factor in place. The problem with this architecture is that the termination condition is not known in advance, making it difficult to decide when to halt the process of finding a solution because there is always room for more refinement. Multiple knowledge sources must be synchronized, which is challenging [57].

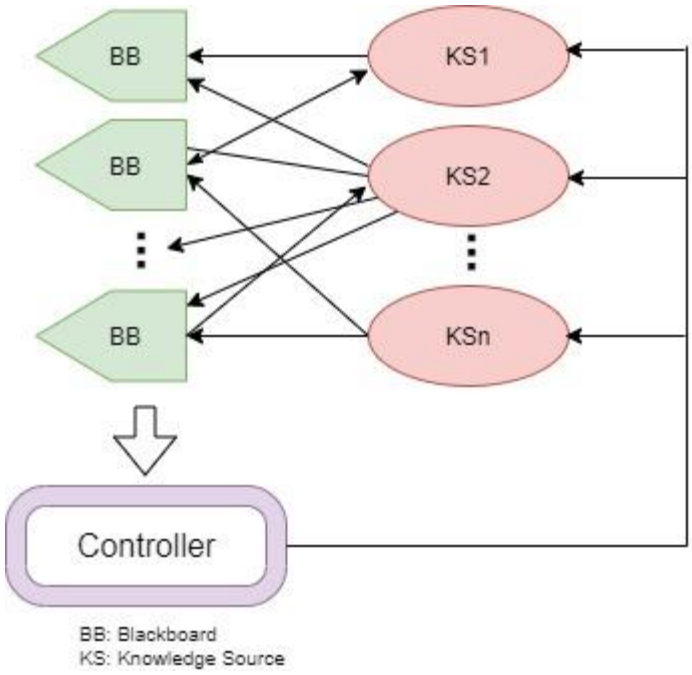

Figure 2: Black board Architecture Style

This diagram describes the working of the Black Board Architecture Style; KS denotes the knowledge Sources of the Blackboard and BB describes the Blackboards and the Controller is the central unit controlling the data moving between KS and BB.

3.1.2. Client–server architecture

The majority of Internet based programs in use today, including those we are using to connect to the Internet, are client-server driven. The entire system is divided into two halves in this instance, in which one acts as a user and the other as a host [58]. Many web-based applications are built on client-server architecture. Figure 3 mainly shows three components.:

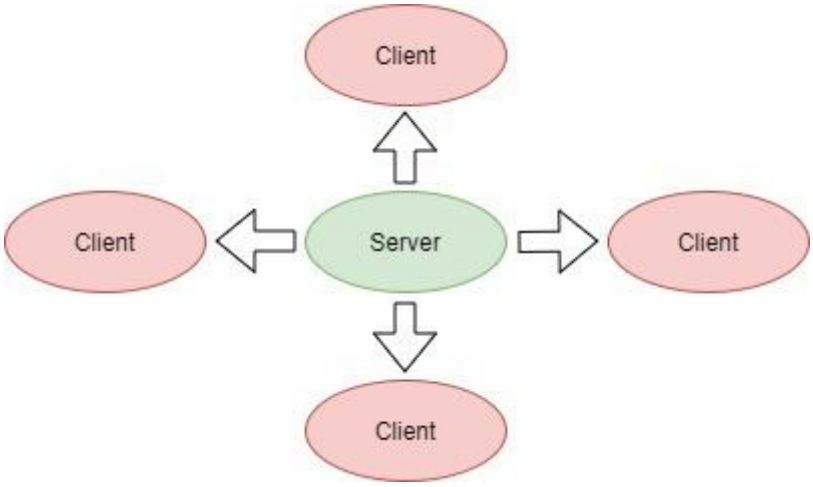

Figure 3. Client Server Architecture Style

This diagram describes the working of the Client Server Architecture Style; Server is the central unit and it process the client's requests and Clients are the entities connected and are the information/service requesters from the Server.

- Client: A service or information requester is referred to as a client. Examples include browsers, chat programs, email clients, etc.

- Server: The client request is received at the host end, do the processing on it, compiles the important information, produces an reply to the client's request, then sends the produced response to the sending node. After sending a request, the service server is ready to take more. Print servers, databases, web-based servers, FTP servers, and chat servers are a few examples.

- Client and server interaction medium: This describes the communication path between servers and clients. A few applications are the intranets, Internet, Bluetooth networks, etc. Figure 3's arrows denote a bidirectional medium of communication. Consider a network of many connected computers where one will function as a print server (where a printer server is mounted) and the rest will serve as clients. The client computer will ask the server computer to print something. The print server will process the query and print the page if it is ready; otherwise, it will put the customers' demands on a waiting list if it is already busy. There are several significant aspects of this architecture:

    - The request's creator will represent the client.

- The request will be handled by the person who serves it.
- The client must know the address of the server to send a request; however, the address of the client is not important for the server.
- To serve the end node, one host can access another host server.
- Clients and servers can switch roles depending on the situation. A machine may occasionally perform both client and servers roles [59].

Client-Server architecture types include:

2-Tier Architecture: There is no middle point or node between the client and the server for communication in this architecture. Although this design offers quick service, it has performance and security flaws. Example: 2-tier client-server architecture is used by Internet Explorer. Three-tier architecture: Between the client and the server, one more node known as the middle tier is located. The middle tier receives the client's request, authenticates it, and approves it before transmitting it to the host [60]. Again, the server generates the response for the middle tier and, after proper verification and validation, passes it to the corresponding client. In cases of high load, the middle tier serves as a load halter and enhances system security.

A higher level of security is achieved by data storage at the host end rather than at the client systems. Using client-server architecture, data accessibility at the central level is made possible. Also included is improved data sharing. Dependency on the server, where the entire system would cease to function if the server went down, could be an issue in this case. Additionally, network congestion may cause the process to lag.

3.1.3. Component-based Architectural Style

The component-based architectural style is built on the concept of issues segmentation. In compliance with the segregation of issues principle, a system is divided into several divisions, each of which addresses a different concern. This method divides the system into a number of physical or logical parts with distinct connections, each of whom

defines a certain feature or fragment of data. In this context, a component could be a web service, a feature, a resource, a package, etc. [61]. Many components distribution is like:

- Fully or partially experienced components: These are parts that are in the organization's library and have been utilized to create a variety of systems.
- Off-the-shelf components: Substances found in a third party's collection.
- New component: Since these elements are not included in either the owner's or any third-party repository, they should be created from scratch.

Since a system is composed of several components, it is simple to incorporate components from one system into another. As a result, it has the capability of reusability [62]. It adheres to the HIGH COHESION method. LOW COUPLING exists between the system's parts. Low coupling indicates that components should be less dependent on one another, and high cohesion indicates that a component performs a single, connected activity. It is simple to add extra functionality or data to a component. It will be simple to identify incorrect component [63] because the system is composed of independent components. It is simple to swap out an outdated module for a new one for system’s maintenance, enhance functionality, or fix errors. Third-party libraries make it simple to locate the necessary component, resulting in quick and simple system development [64]. The idea of reusability would not exist if new technology were adopted. Thus, the primary benefit of this architecture is useless in this situation. There will always be disagreements over "system evolution," "compatibility," and migration.

Figure 4 shows 5 components : **User Interface**: For the user to interact with the system; **Notification**: In order to facilitate client communication with the system and inform clients of their responses, 3. **Order Management**: It handles customer orders. 4. **Accounting**: payment related tasks are neglected here. 5. **Persistence layer**: Data or information repository Component models: They outline a component's implementation, documentation, and deployment. Common Object Request Broker Architecture (CORBA) Component Model and Enterprise JavaBeans (EJB) model [65] are two

examples of common component models. To visualize system in low level abstracted form, we can further subdivide the components into subcomponents.

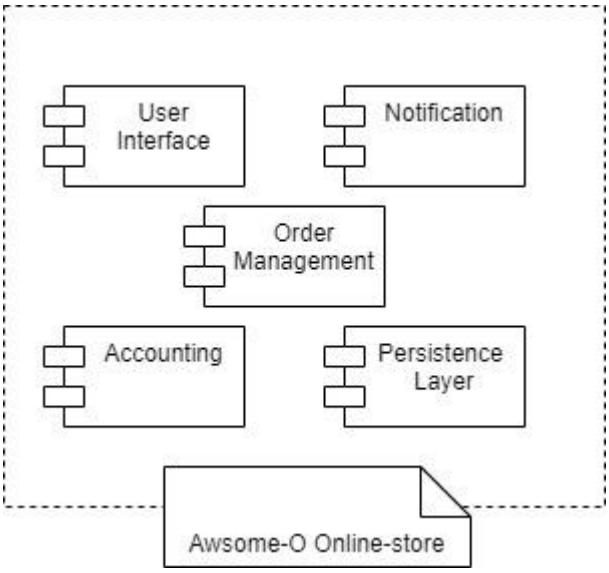

Figure 4. Component Based Architecture Style
This diagram describes the working of the Component Based Architecture Style; the system is divided into several logical or physical components with clearly defined interfaces, each of which defines a particular functionality or piece of information and is handling a distinct concern.

3.1.4. Event driven Architecture

Event driven architecture represents all aspects of the initiation, execution, surveillance, and end of events that may occur in a system. Here, the word "event" refers to a "status change." The condition of an object can be tracked using controllers, sensors, and other sensing devices [66]. For example, in the case, the electric light changes from "OFF" to "ON" when button is pressed.

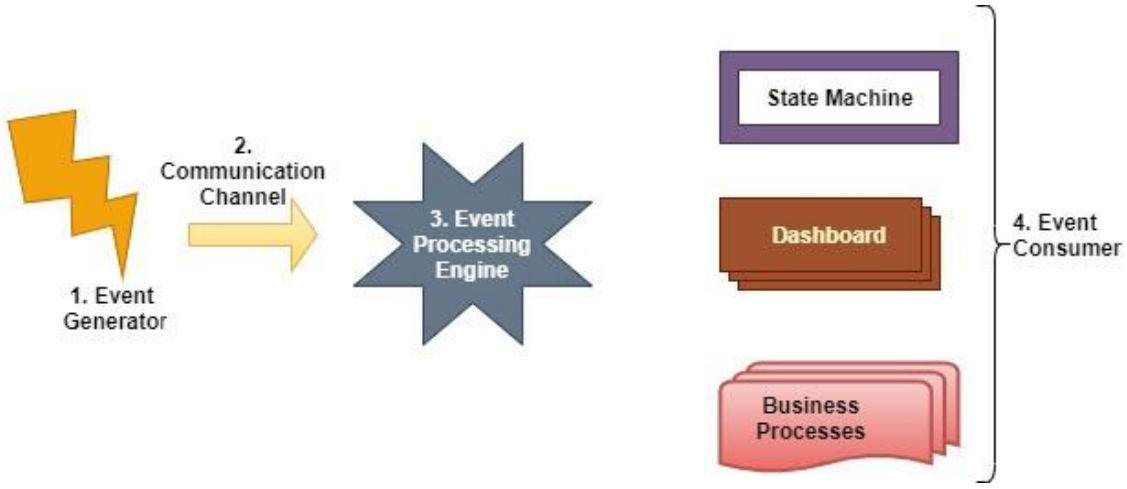

Figure 5. Event Driven Architecture Style
This diagram describes the working of the Event Driven Architecture Style; The origination, processing, monitoring, and termination of events that can happen in a system are all represented by event driven architecture.

Four logical layers make up event-driven architecture: 1. **Event generator**: The event source is in this layer, creates the events. The event cause could be a click on mouse, key pressed on keyboard, email sent to the client, etc. **2.Event Channel**: The event must be transferred to the customer end from the source, such as an event generator or sink. Events are first stored in a queue after which they are eventually sent to the sink using an event channel. Event channels might be input files in XML format or TCP/IP connections [67]. **3. Event processing engine**: This component deals with event processing and generating the particular response to the event. 4. **Event Consumer** that occurs later: Here, the effects of events are displayed. Sending mail without a topic, for instance, triggers the warning message. Emailing is therefore an event, and the error message shows the result of the event. It is ideal for creating interactive systems (e.g., GUI). As a result, it is excellent for most applications used today. It could take a while to process an event at times, which can make it slow. It leads to a sophisticated system [68].

3.1.5. Pipes and filter architecture

Here Filters are employed as processing components, and pipes are utilized to connect them so that the output of one filter serves as the input for the next. There might be some intervening components that function as a buffer to control the information flow between

two filters. Figure 5 illustrates a unidirectional software architecture. Examples of filters are threads and processes. The UNIX operating system uses this style of architecture. The filters can all be spread across many systems and run simultaneously [69]. The source is connected to the first filter, while the sink is connected to the last filter. It is important to appropriately connect the pipes so that the output of one filter enters the corresponding filter. Independent filters can operate concurrently, but those that are receiving input from a prior filter must wait to get it. Input into the system is provided by a pump or source, and this input travels through a pipe to the linked filter. The filter processes the information and sends the output to the next filter for additional processing. At the same time, the previous filter accepts new input and processes it, reducing processing delays overall. By eliminating the processing stalls, the system's performance is enhanced. Thus, such a system provides greater throughput. Since filters don't exchange their state, they are only loosely coupled to one another. Increased maintainability, or the fact that this system has more components, necessitates increased component management and maintenance. Error handling and solving in this technique is difficult since states are not shared [70].

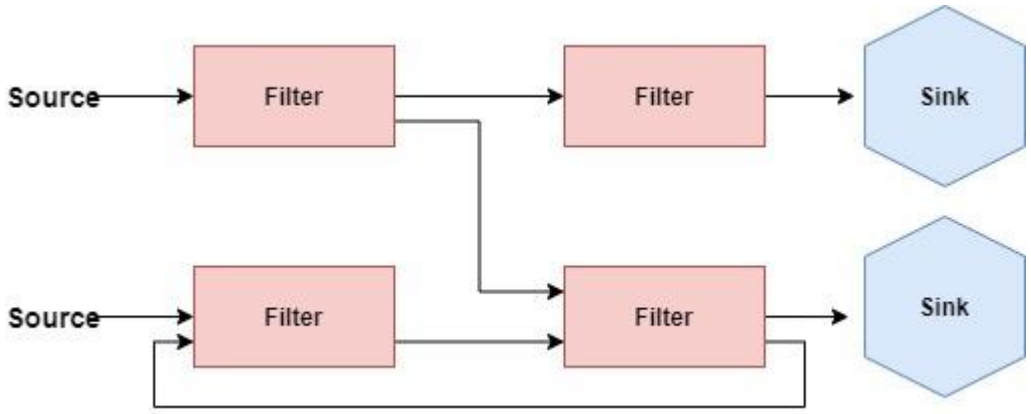

Figure 6. Pipe and Filter Architecture Style
This diagram describes the working of the Pipe and Filter Architecture Style; Filters are employed as processing components, and pipes are utilized to connect them so that the output of one filter serves as the input for the next.

3.1.6. Layered Architectural Style

The duties of the software are divided into numerous loosely linked layers via layered architecture. Based on separation of concerns, a system is layered. Layers can talk to one another via clearly defined interfaces. Layers can be dispersed over multiple computers or reside on a single machine because they are loosely connected. A software system is typically broken down into three tiers: User interface functionalities are included in the presentation layer. The user can communicate with the system through this layer. Business Layer: The system's business logic is incorporated into this layer. Data related services, services related to the networking, and other services related to the infrastructure are all included in the infrastructure layer. Through the service interface, external clients or applications can directly use the business layer's services. Example: The TCP/IP and OSI (Open system interconnect) models provide the best illustration of layered architecture. In the OSI model, the system's services are divided into seven distinct levels, as seen in figure 6. Regardless of internal variances, the OSI model depicts how systems communicate with one another in a network. The topology (mesh star, hub, bus, etc.) and configuration (point to point, multi-point) of the network are represented by the physical layer [71]. Bits are used in this instance to represent data. Local data delivery from node to node is handled by the data link layer. It controls flow and local errors. Frames are used to represent the data in this layer. Delivering data from host to host is the responsibility of the network layer. It carries out packet routing from one node to the next. Packets are used to represent data in this layer. The data delivery from process to process is handled by the Transport Layer. It controls flow and errors globally. This layer's data is provided as a segment or datagram. The management of a user's session falls under the purview of the session layer. With the use of checkpoints, it controls and synchronizes dialogue. The presentation layer oversees the data's encryption, translation, and compression. Application layer functions as user interface layer [71] by including user services. Thus, the presentation layer is represented by the data link layer, while the infrastructure layer is represented by the application layer, and the business layer is represented by all other levels between these two. One may easily scale and maintain the system while working on one of the system's many layers at once. In this case, users communicate with components through layers rather than simply calling

them. It can impede all system-wide communication between the various parts. It's also challenging to determine a system's precise layer count.

3.2. Software Architecture Design Process:

The transformation of the customer requirements as outlined in the SRS documents into a format that can be implemented using a programming language is the focus of the software development design phase. The three layers of design phases that make up the software design process are as follows:

1. Interface Design
2. Architectural Design
3. Detailed Design

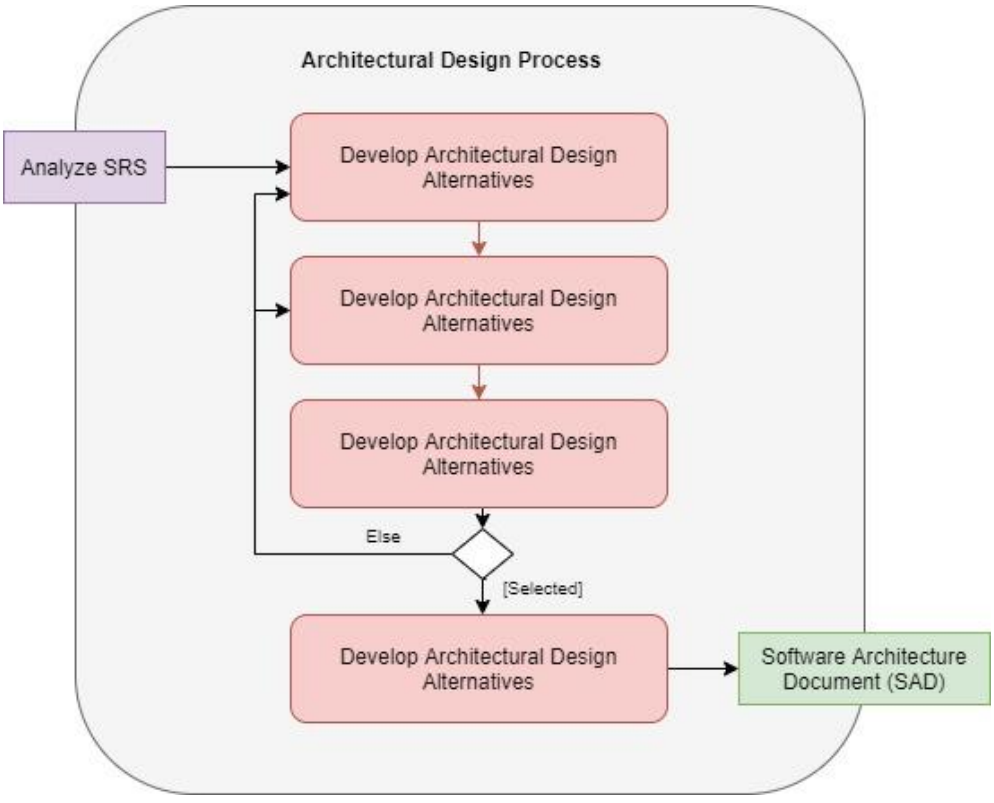

Figure 7. Architecture Design Process

This diagram shows the overall workflow of the creation of Software Architecture Design (SAD) document from Software Requirement Specification (SRS) document. SRS Document is analyzed and on the basis of the Non-functional Requirements some architecture style is proposed.

3.3. Quality Attributes:

To reach the system's functional specifications, users and other stakeholders have concerns that are expressed in the quality attributes of application programmes [72]. Software quality models among them that has received most attention and covers more facets of quality of software is the ISO/IEC quality model. The International Organization for Standardization (ISO) developed this model in 1991 in response to the software industry's need for a standardized method of software evaluation. It was changed once more in 2001 [73]. As seen in Fig. 1 of the ISO/IEC model 1. Six primary quality parameter, several supplementary parameters are connected to it, are used to describe the quality of software. The following are these qualities:

- **Functionality:**

  Capability is the ability of the software to meet functional requirements in a particular circumstance. The suitability, accuracy, interoperability, and security of this feature are its sub features.

- **Reliability:**

  Reliability, which has the sub features of maturity, fault tolerance, and recoverability, is the capacity of a software product to maintain a specific level of performance when used in a particular environment.

- **Usability:**

  It is a feature that describes software products' ease of use, how appealing it is, and how useful it is under usage circumstances. Sub features of this characteristic include operability, learnability, and understandability.

- **Efficiency:**

  Efficiency shows the software's capacity to deliver proper performance, linked to the number of resources used in specific situations [73]. We can also say, efficiency is the amount of time that a system must take to respond to a provided number of events in

a given period of time. It demonstrates how effectively system components interact with one another. It aims to shorten system response times and times wasted waiting. Utilizing resources effectively and managing your time are efficiency's sub features.

- **Maintainability:**

The capacity of a software product to implement modifications and fluctuations. These modifications affect testability, stability, variability, and analysis capability.

- **Portability:**

The capability of the system to run under different computing environments, including hardware, software, and structural environments, is expressed by the attribute "portability." Sub-features should be examined to assess portability, adaptation, installation, and coexistence.

3.4. Software Requirement Specification (SRS) Document:

A Software system detailed explanation that has to be created is called a software requirements specification (SRS). The business constraints specification serves as its model (CONOPS). The software requirements specification outlines both functional and non-functional needs. It may also contain a list of use cases that illustrate the ideal user interactions that the product must enable.

The software requirements specification serves as the basis for a contract between customers and vendors or contractors regarding how the software product will function. Before the more detailed system design stages, software requirements specification performs a comprehensive review of the requirements with the intention of reducing later redesign. Moreover, it must offer a solid organization for estimating product prices, risks, and timelines. Software project failure can be avoided when software requirements specifications are used properly. Figure 7 shows the overall architecture of the SRS document. We are using the PURE dataset for our Project. It contains 79 SRS documents.

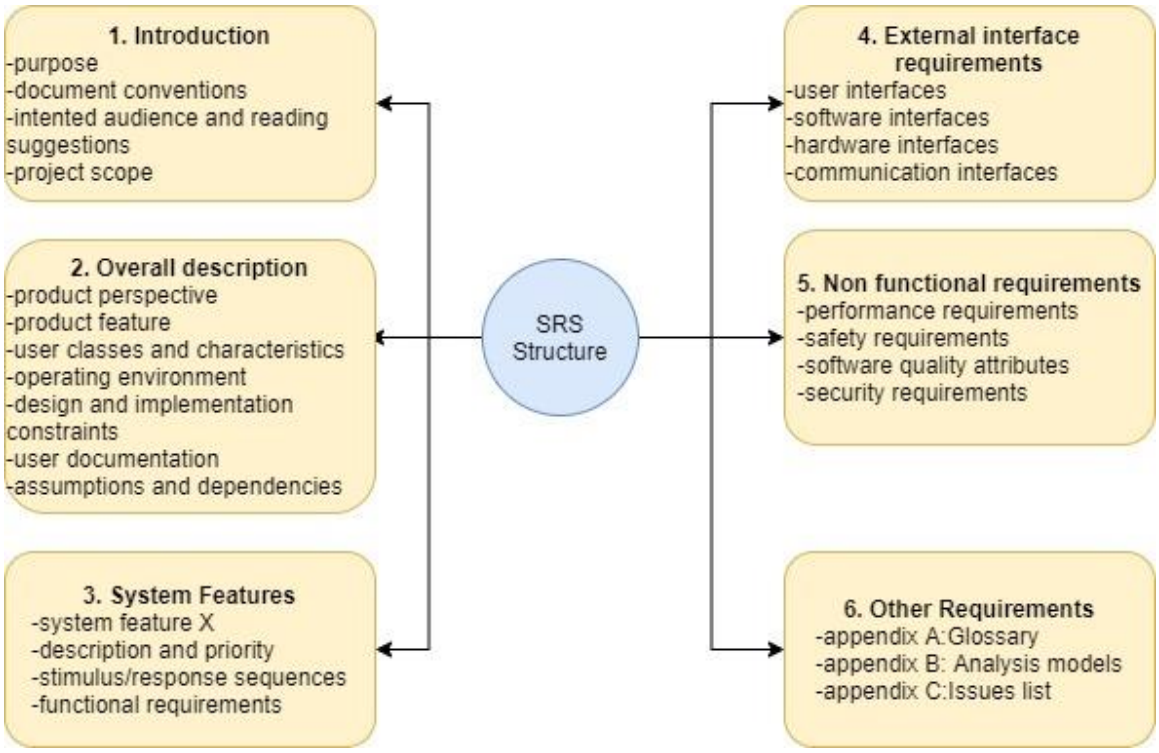

Figure 8. SRS Document Structure

It describes the overall different chapters and their sub attributes of SRS Documents. It includes Introduction and project description; System Features are defined; Functional and Non-functional requirements of the system is also defined in detail.

Chapter 4

Natural Language Processing Techniques and Tool

Initially a brief overview of the Natural Language Processing will be provided and further generate a review for the techniques used for the unstructured data extraction; NLP for UML diagrams generation; and NLP usage in architecture style prediction.

4.1 Natural Language Processing Overview

The Natural Language Processing (NLP) discipline uses computational methods to learn, comprehend, and create content that is understandable by humans [83]. According to Liddy et al., NLP is a computational method for analyzing and representing texts at various linguistic levels to incorporate human-like language processing for a variety of activities and applications [84].

The research in information retrieval aims to retrieve the desired information from data sources by developing different models and algorithms. The information extracted is usually in the form of structured or unstructured text written in Natural Language (NL). The orthodox problem of retrieving information is called ad-hoc retrieving problem. In this whenever a user demands any information, they request it through a query, and it will return a list of related documents. Matching systems return documents exactly mapping to the query statement. Manning and Schütze [80] claims that results for the query requested against a huge and diverse corpus will be either empty or huge. So recently, documents ranking according to the query mapping approach is focused. It will retrieve the most relevant results. Following are the NLP stages as described by the Manning and Schütze [80]:

4.1.1. Tokenization:

Tokenization is the conversion of the sentences and words, into tokens of words and characters respectively. It is achieved by removing the punctuations, capitals, parenthesis etc. So, we can say that each token is a word but definition of word is not literal one. It could be a alphanumeric characters enclosed in spaces.There are different ways to

remove the special chracters like hyphens, inverted commas and punctuations.Commas and colons are easy to remove as they are not connected with words but to remove apostrophies and hyphens is a confusing task, for example the girl's doesn't clears whether its girl is or girl has.

Similarly, splitting the hyphenated words is also debatable as some words like E-mail if we remove hyphen then it will be considered as a single word, but some words after removing hyphen should be considered separately, as for text-based medium they are used as hyphenated pre-modifiers. Different techniques are used to split data and it is completely dependent on the data type to be tokenized.

4.1.2. Stemming:

Stemming is extracting the root form of a word. Writing style of the words can be different, but each grammatically diferent form has same core meaning. During the fixation phase the different lexical forms of the words are replaced with the stemmed word, for example, the words writing and written are stemmed as write. Also the Verbs are stemmed in such a way that they transformed into the first form of verb like, be will be the stemmed form of being and was.

4.1.3. Stop Words Removal:

Certain words that doesn't carry any meaning or useful information are not helpful in finding the similarity index, like the words this, that, the, when are the stop words. These are important for sentence semantics but are not helpful in relating the content to the query, as they are present in all text documents with almost same frequency.It is important to remove these words as they are important for the similarity analysis. Mostly stop words are conjunctions, pronouns or prepositions.

A list of stop words is provided to the system to be removed from the text. The words that match the stop words' list are removed and only the specific words are kept [80].

To reduce the impact level of the words on the similarity index to documents, stop words should be removed from the template document. Another approach that can be used to reduce the impact of stop words is by using inverse term frequency method, it will invert the weightage of the words in the whole corpus [81]. So most frequently occurring words will have low weightage and less frequent words will have high weightage and more contribution in similarity index estimation.

4.1.4. The Vector Space Model:

Different words are displayed using this paradigm in multidimensional vector space. Each word in vector space corresponds to a specific dimension.

The words are then represented in a multi-dimensional vector space model as the next step. A term is associated with each dimension of the space. According on how frequently a word appears in the text, it will be positioned along each axis in this area. The distances in this vector space are then used to calculate how similar two texts are to one another. The dimensions need not be strictly linear, such as a straightforward word count. A word that appears three times is likely more significant to the text's substance than a term that appears just once but is not three times as significant. The phrase frequency must be attenuated as a result. Using a weighted scale, as indicated in equation (1), is a typical strategy.

$$weight=1+\log(frequency) \qquad (1)$$

It is possible to expand this analysis to word pairs or word triples. Although this makes computations more complex, it has been successful in other instances [81].

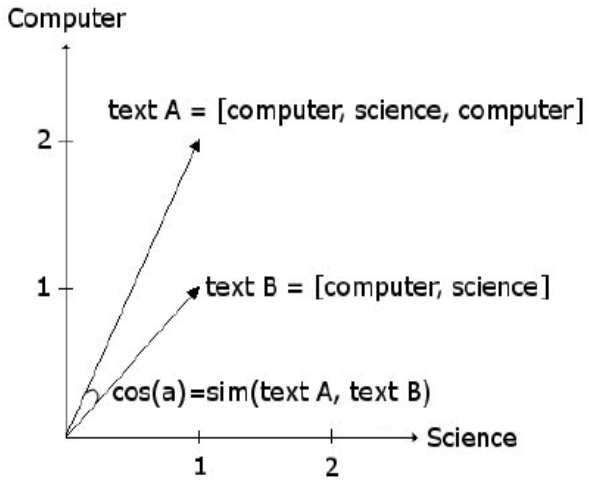

Figure 9. Using the Cosine measure and the vector space model, compare the two texts computer, science, computer and computer, science

4.1.5. Similarity Measures:

The vector space model is used to determine how similar two texts are. Cosine, Dice, and Jaccard are the three most frequently used measurements [80]. The choice of a certain vector-similarity measure for a given application is up to the user, says Salton, and is not dictated by any theoretical considerations. [82].This model is used to show different words in multi-dimentional vector space. Each word is corresponded to a certain dimention in vector space.

To take the length of the vectors into account, all three measures have been normalized. *Fig. 7* illustrates the calculation of a similarity measure graphically.

4.2. NLP techniques for Unstructured Data mining

There are a lot of NLP techniques used for the structuring and classification of the data. Here are some techniques discussed for data structuring:

4.2.1. Tokenization:

It is a technique in which whole documents are partitioned into tokens based on the words, sentences, spaces, and full stops.

4.2.2. Part-of-Speech (POS) Tagging:

This technique makes tokenization based on the parts of speech like verbs, nouns, adjectives, adverbs etc. Commonly they work statistically, they are trained based on the manually designed corpuses for the prediction of POS tags.

4.2.3. *Shallow Parsing:*

It is a technique in which nouns and verbs are separated out from the sentence. This creates the chunks of sentences and known as Noun Chunking and Verb Chunking. For example, in sentence Messages are delivered to the users, a shallow parser will identify the Messages and the user as Noun Phrases and are delivered as Verb Phrase.

4.2.4. *Gazetteer:*

This will search for terms' occurrences that is defined in the list of terms. This scenario is basically to check the vague terms from the list of terms.

4.2.5. *JAPE Rules:*

This technology will define the rules like the regular expressions over token and other element in the text. It will detect the elements that coordinate with the rules.

The JAPE grammar are the rules like the regular expressions. Reporting of these rules can be rather long to report. Some papers have described the JAPE grammar rules.

4.3. NLP tools for Unstructured Data mining

There are number of NLP tools used for the mining of unstructured data from different repositories. Here are some tools and their functionalities listed below:

4.3.1. Stanford Core NLP:

It is the analysis tool for the NL. It enables us to do number of different functionalities like POS tagging of sentences, pattern learning using bootstrap, co reference tenacity, Named entity recognition, etc. It gives assistance for different programming languages and its interfaces are designed for even the recent emerging ones [74].

4.3.2. *Apache OpenNLP:*

It is a Machine Learning toolkit that works using Java language. It processes the free text in Natural Language. It helps in phrase and sentence identification, tokenizing and de-tokenizing, NER, categorization of documents, POS tagging and co-reference resolution. It also assists in maximum entropy ML [75].

4.3.3. *Python NLTK:*

NLTK is python-based package that helps in implementing the different data structure and algorithms. It enables us to tokenize, classify, do stemming, chunking the text, POS tagging, sentiment analysis and reasoning, etc [76].

4.3.4. *GATE*:

It is a Java based tool to analyze the text and is also an open source. It assists in doing number of different tasks like tokenizing, sentence splitting, POS tagging, NER, semantic tagging and anaphora resolution, etc. [77].

4.3.5. SpaCy:

It's also an open-sourced Natural Language Processing based tool. It enables us to perform multiple tasks like dependency parsing, rule-based matching, similarity calculation, tokenizing, POS tagging, lemmatizing and NER etc [78].

4.3.6. EmoTxt:

It’s a tool that helps in the recognition of emotions. It extracts the emotions that are positive and negative in nature. It also extracts the emotions from the text [79].

4.3.7. MALLET:

It’s a Java based toolkit for statistical analysis using NLP. It has some functionalities like text categorization or classification, topic modelling and sequence tagging [80].

4.3.8. *KERAS:*

It is a tool that works using CNTK, TensorFlow and Theano at the backend for deep learning. It assists in implementing the neural network, convolutional network and recurrent networks and it also assists in transfer learning for NLP [81].

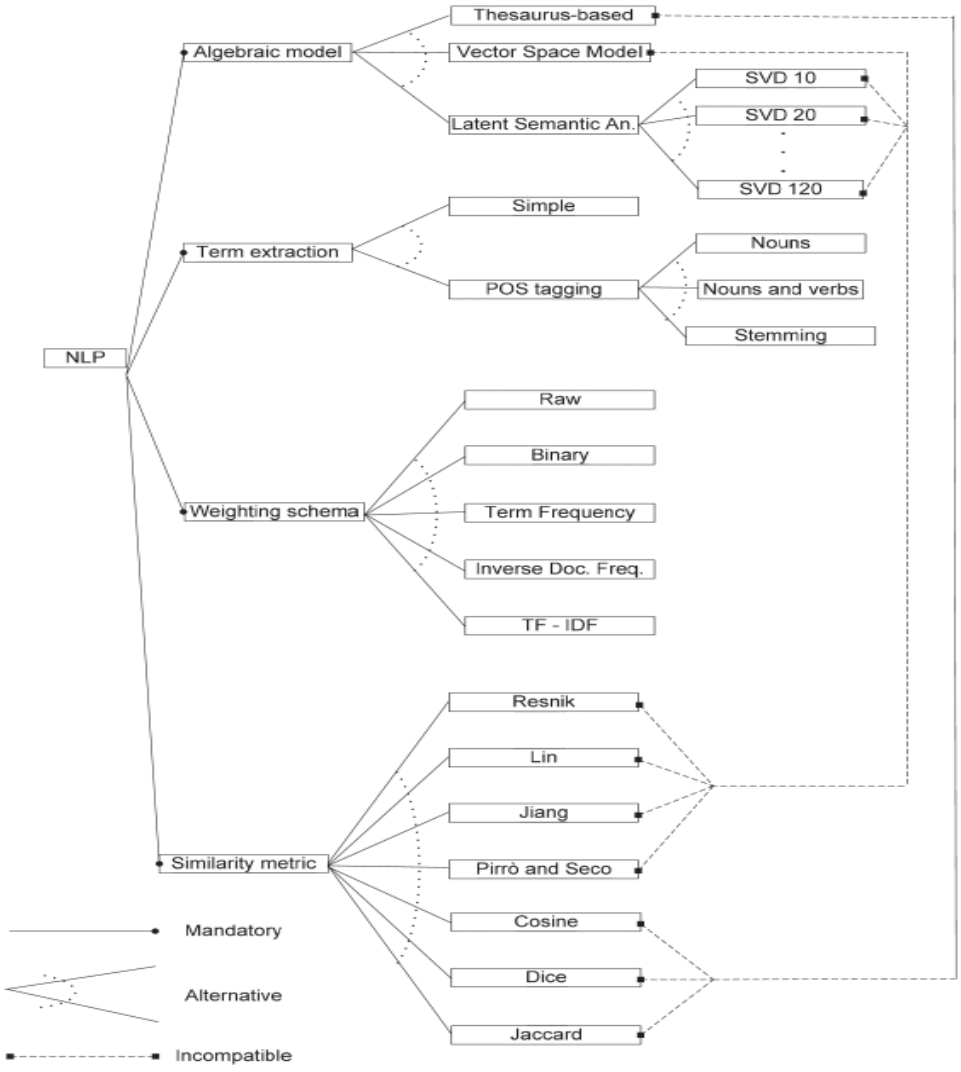

Figure 10. Natural Language Processing (NLP) Techniques Classification

This flow diagram shows the different NLP techniques and their applicability with respect to different approaches.

Chapter 5

Research Methodology

For our research, we are considering 9 architectural styles. There are several different programming paradigms, including batch-sequential (BS), pipe-and-filter (PF), virtual machine (VM), client-server (CS), publish-subscriber (PS), event-based (EB), peer-to-peer (PP), C2 (Component and connector), and CORBA (Common Object Request Broker Architecture). For our investigation, we are using a table as a guide. To categorize various architectural styles, Dwivedi et al. used 8 quality attributes, such as efficiency, complexity, scalability, heterogeneity, adaptability, portability, dependability, and security. These descriptions are shown in table 3.

- The sign (++) indicated in table 3 depicts that a certain quality feature is performed exceptionally effectively by a certain architectural style.
- The plus sign (+) denotes that a particular quality attribute is supported by a style in some way.
- A style that has no effect on a quality attribute is denoted by the number "0".
- The letter "-" stands for the style's detrimental effects on some quality attribute criteria.

These annotations aid in a clearer understanding of the classification and assessment of various architectural styles [85]. Each application has different requirements for these quality parameters' supportability. However, these kinds of descriptions of many architectural styles aid an architect in the high-level design of a software.

Table 3. Architecture Styles and Quality Attributes Categorization and Evaluation

QPs / Styles	BS	PF	VM	CS	PS	EB	PP	C2	CORBA
Efficiency	0	0	-	-	0	0	+	+	-
Complexity	0	0	0	0	+	+	++	+	++
Scalability	0	+	+	+	0	+	+	+	0
Heterogeneity	-	-	+	-	+	+	0	++	++
Adaptability	0	-	0	0	+	0	0	+	++
Portability	0	0	++	0	++	+	0	++	+
Reliability	0	0	0	-	-	-	++	+	0
Security	0	0	0	++	+	0	+	0	-

5.1. Proposed Methodology

The Automatic Architecture Style Predictor (AASP) is a multiple-criteria decision-making process that chooses the most suitable choice among all options [90] based on a number of factors that may be dissimilar or even at odds with one another. This approach involves manually breaking down the problem into its component parts, creating a hierarchical tree, performing a comparison of the particular categorization criteria at the similar level, and then assigning a number to each solution to rank the solutions and enable the best one to be chosen [89]. According to AASP, these are the crucial phases to solving complicated, multi-criteria problems:

1. Creating a hierarchical structure for the definition of the decision-making criteria, with the aim at the top, the requirements and associated requirements in the middle, and the potential solutions and possibilities at the bottom.
2. Establishing weights for substitutions, associated requirements, and evaluating criteria based on the respective relevance of each component from its superior stage.
3. Rank the top three best suitable Architectural Styles.

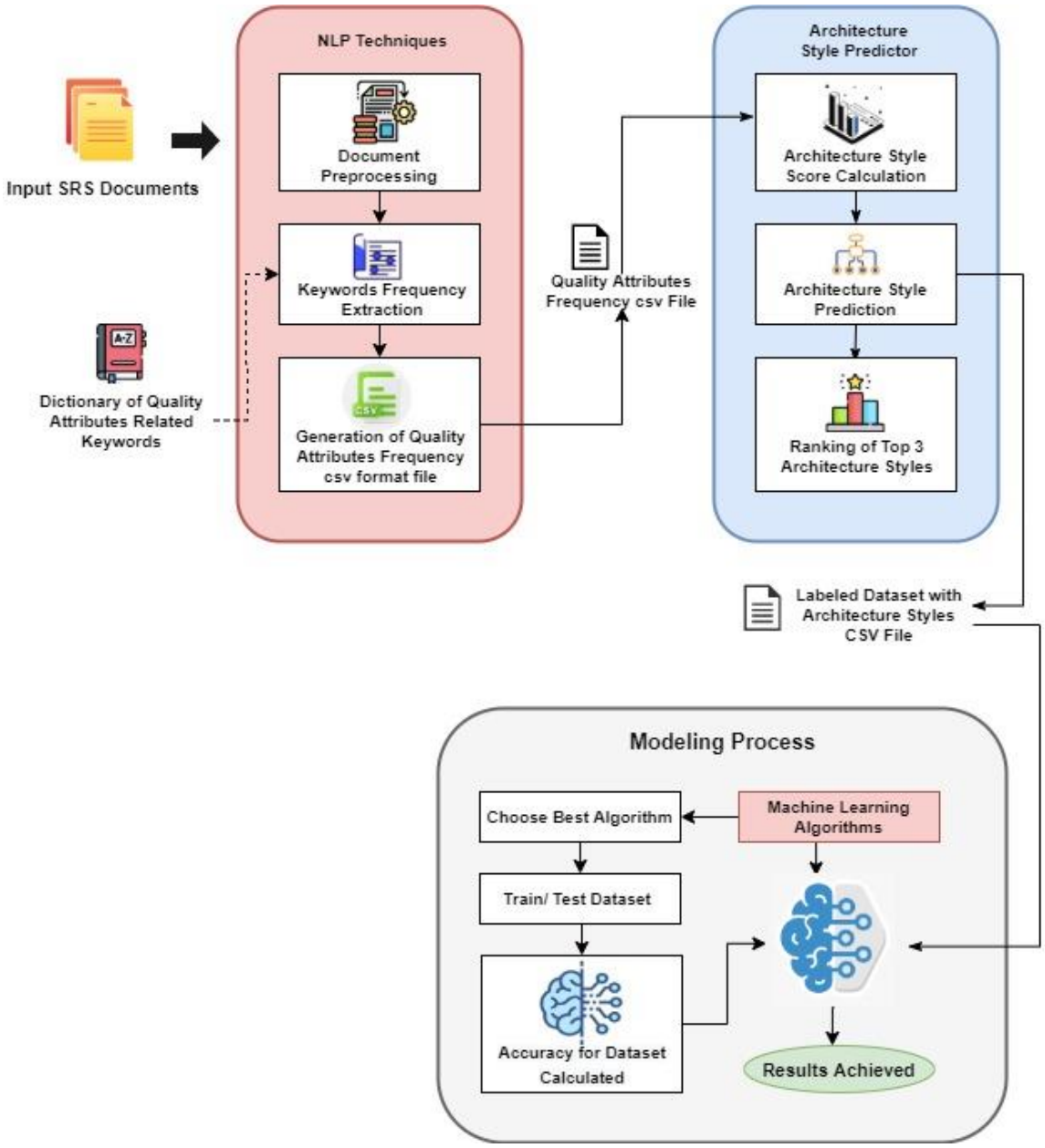

Figure 11. Automatic Architecture Style Predictor (AASP) Architecture Diagram

This Diagram depicts the overall working of the Automatic Architecture Style Predictor. A. NLP Techniques: When the document is fed into the system it will require some preprocessing and features extraction; B. Architecture Style Predictor: this phase involves the architecture styles scores calculation and selection based on quality attributes categorization; C. Modeling Process involves different Machine Learning Processes that will be used predict the accuracy of dataset through training and testing of dataset.

Despite AASP, uncertainty in the ranking of alternatives is caused by inefficiency in the management of quality based benchmarks set, a lack of precise standards for measuring and recording the statistical measures of the quality based benchmark, and vagueness and inconsistency in comparative assessments [87].As a solution to this problem, a combination of Automatic Architecture Style Predictor AASP Natural Language Processing is used to extract the keywords related to the quality attributes from the document. To improve the process's realism-adjustment, account for concerns with ambiguity and inconsistency, and ultimately improve the accuracy of ranked options [88]. In this technique, normalization technique is used to extract the proportional weights for each Architecture Style were used to compute relative weight. The different stages of AASP based on the extensive analysis method are as following [86]:

5.2. Creation of the Architectural Quality attributes Pairwise Comparison Matrix:

A comparison matrix is defined according to the quality attributes categorization in Table 1, and by decision-makers. The normalization is done by calculating the "-" and "+" signs and divide the "-1*n" or "1*n" (where n is the number of occurrences of "+" or "–" against that attribute) with the total number of occurrences of the sign respectively. The keywords used for Quality attributes are described in Table 4.

Table 4. Quality Attributes Keywords

Quality Attributes	Keywords
Security	Security, Confidential, Integrity, Non-Repudiation, Accountability, Authenticity, Compliance, Secure
Reliability	Availability, Fault Tolerance, Recoverability, Available
Efficiency	Efficiency, Time Behavior, Resource, Utilization, Speed, Effort, Productivity, Performance, Throughput, Productivity, Effectiveness
Complexity	Complication, Intricacy, Ramification, Convolution,

	Elaboration, Entanglement, Involvement, Multiplicity, Complexness, Complicacy, Complicatedness, Elaborateness, Intricate, Intricateness, Involution, Knottiness, Sophistication
Scalability	Expandability, Expandable, Ease Of Use, Dependability, Dependable, Reliability, Reliable, Extensible, Scalability, Workload, Competitiveness, Endurance, Scaling, Processing Time, Manageability, Portability, Robustness, Extensibility, Connectivity And Functionality, Scalable
Heterogeneity	Operatable, Maintainable, Technical Accessibility, Modularity, Learnability, Attractiveness, Recognizability, Appropriateness, Reusable, Analyzable, Distributed System, Operational
Adaptability	Elasticity, Flexibility, Limberness, Resilience, Workability, Workableness, Ductility, Pliability, Pliableness, Pliancy, Pliantness, Suppleness, Malleability, Plasticity, Ease Of Use, User Friendly
Portability	Transferability, Transferable, Installability, Installable, Adoptability, Adoptable, Interoperability, Interoperable, Coexistence, Compatible, Replaceability, Replaceable, Flexibility, Flexible, Maneuverability, Motility, Movability, Adjustability, Adjustable, Moveable, Transportable, Moveableness, Transportability, Portable

By considering the following short forms of Frequency of Quality Attributes:

- Frequency of efficiency keywords : Fe
- Frequency of heterogeneity keywords : Fh

- Frequency of portability keywords: Fp
- Frequency of reliability keywords : Fr
- Frequency of scalability keywords : Fsca
- Frequency of Security keywords : Fsec
- Frequency of adaptability keywords : Fa
- Frequency of complexity keywords : Fc

5.3. Calculation of Architecture Styles Weights:

By considering the specified criterion, estimate the weights for each requirement and each its alternatives. Summation is computed for each row of the comparison matrix considering the following equations for the calculation of Architecture Styles weights:

BS = Fa*0 + Fc*0 + Fe*0 + Fh*(-1) + Fp*0 + Fr *0 + Fsca*0 + Fsec*0

PF = Fa*(-0.5) + Fc*0 + Fe*0 + Fh*(-0.5) + Fp*0 + Fr*0 + Fsca*1 + Fsec*0

VM = Fa*0 + Fc*0 + Fe*(-1) + Fh*0.25 + Fp*1 + Fr*0 + Fsca*0.25 + Fsec*0

CS = Fa*0 + Fc*0 + Fe*(-1/3) + Fh*(-1/3) + Fp*0 + Fr*(-1/3) + Fsca* (1/3) + Fsec*(2/3)

PS = Fa*(1/6) + Fc*(1/6) + Fe*0 + Fh*(1/6) + Fp*3 + Fr*(-1) + Fsca*0 + Fsec*(1/6)

EB = Fa*0 + Fc*0.25 + Fe*0 + Fh*0.25 + Fp*0.25 + Fr*(-1) + Fsca*0.25 + Fsec*0

PP = Fa*0 + Fc*(2/7) + Fe*(1/7) + Fh*0 + Fp*0 + Fr*(2/7) + Fsca*(1/7) + Fsec*(1/7)

C2 = Fa*(1/9) + Fc*(1/9) + Fe*(1/9) + Fh*(2/9) + Fp*(2/9) + Fr*(1/9) + Fsca*(1/9) + Fsec*0

CORBA = Fa*(2/7) + Fc*(2/7) + Fe*(-0.5) + Fh*(2/7) + Fp*(1/7) + Fr*0 + Fsca*0 + Fsec*(-0.5)

The normalized weights calculated for all the architecture styles are compared and the highest among them is chosen as the top priority architecture style. Then the ranking of top three Architecture Styles is done using those weights calculated.

Chapter 6

Results and Analysis

6.1. Decision Tree

Instances are classified using Decision Trees (DT), which sort instances according to feature values. In a decision tree, each node represents a feature in an instance that needs to be classified, and each branch represents a possible value for the node. Beginning at the root node, instances are categorized and arranged according to the values of their features [94]. Observations about an item are mapped to conclusions about the item's target value using a decision tree as a predictive model in decision tree learning, which is used in data mining and machine learning. Classification trees or regression trees are more evocative names for such tree structures [98]. To evaluate the performance of decision trees while they are pruned using a validation set, decision tree classifiers frequently use post-pruning approaches. Any node may be deleted and have the most prevalent class of the sorted training instances allocated to it [94].

Accuracy Achieved = 70.83

6.2. SVM

These supervised machine learning methods are the latest [99].

Support Vector Machine (SVM) models and traditional multilayer perceptron neural networks have a strong relationship.

The idea of a "margin"—either side of a hyperplane separating two data classes—is central to SVMs. It has been demonstrated that maximizing the margin lowers the upper bound on the expected generalization error by establishing the greatest distance between the separating hyperplane and the instances on either side of it [94]. Figure 12 shows the confusion matrix.

Accuracy Achieved = 68.75

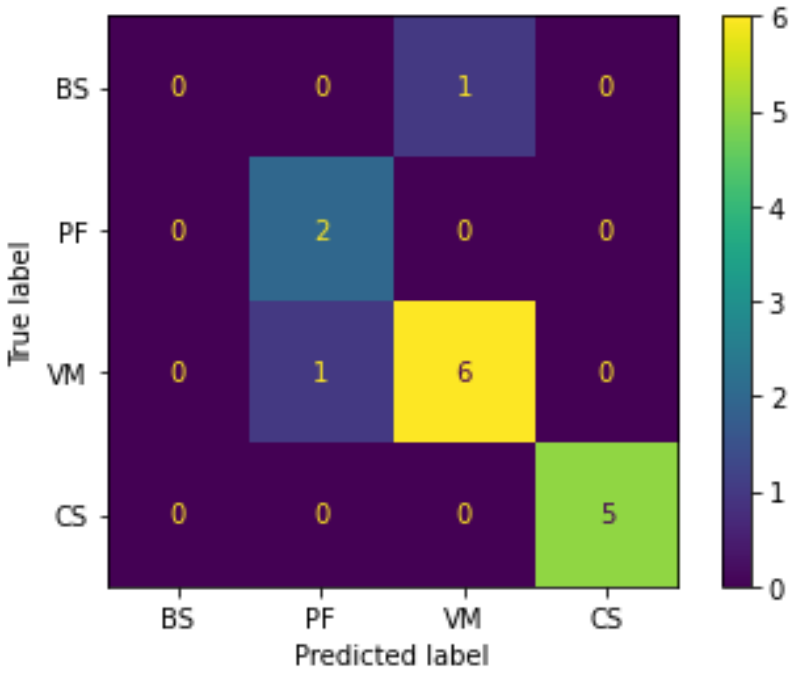

Figure 12. SVM Confusion Matrix

6.3. Random Forest

The generalization error for forests converges a.s. to a limit as the number of trees in the forest increases. Random forests are a combination of tree predictors where each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest. The strength of each individual tree in the forest and the correlation between them determine the generalization error of a forest of tree classifiers. Each node is split using a random selection of features, which results in error rates that are comparable to Adaboost's but more resilient to noise. Internal estimates keep track of inaccuracy, strength, and correlation; they are used to demonstrate how the splitting process responds to an increase in the number of features. Internal estimations are another method for gauging variable significance. Regression can also use these concepts.

Accuracy Achieved = 87.5

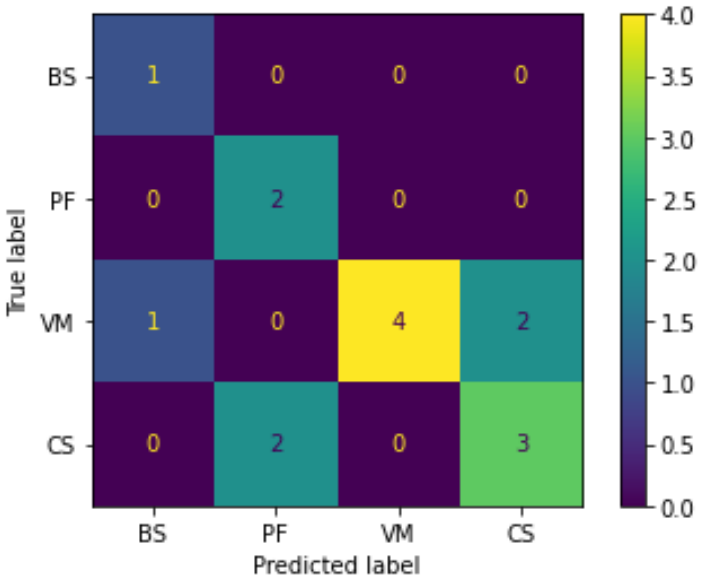

Figure 13. Random Forest Confusion Matrix

6.4. Naive Bayesian

There is a bold claim of interdependence between child nodes and their parents in these Bayesian networks, which are composed of directed acyclic graphs with an only parent (indicating the undiscovered node) and multiple children (matching to discovered nodes) [92].

As a result, estimating is the foundation of the model (Naive Bayes) [97]. When compared to other, more complicated learning algorithms, Bayes classifiers typically perform less accurately. On standard datasets, [91] conducted a comprehensive evaluation of the naive Bayes classifier against state-of-the-art decision tree, specific example based learning, and rule based techniques and discovered that it was hardly better than the other learning schemes, especially when working with datasets with heavy component correlations. The averaged one-dependence estimators were used to solve the attribute independence problem with the Bayes classifier [93].

Accuracy Achieved = 66.66

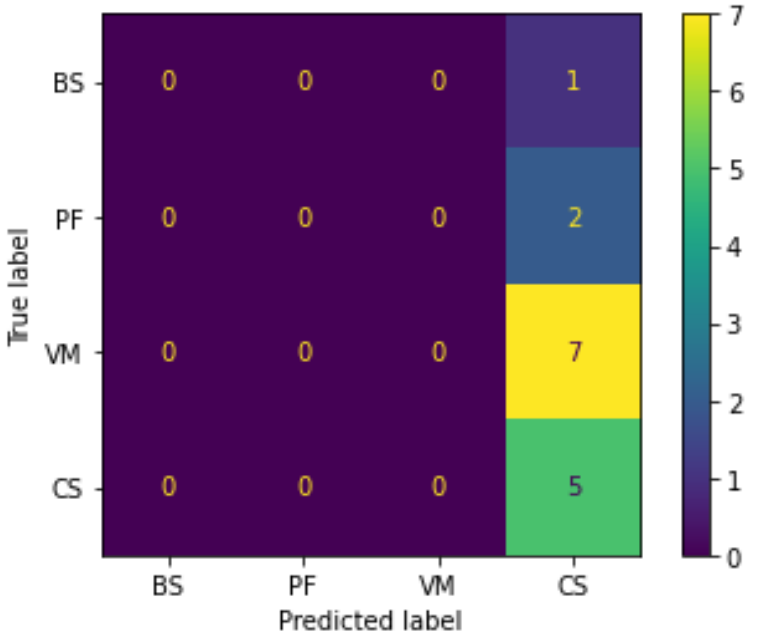

Figure 14. Naïve Bayes Confusion Matrix

6.5. KNN

K-Nearest Neighbor is one of the most basic supervised learning-based machine learning techniques. The K-NN method places the new instance in the class that resembles the preexisting classifications the most, presuming that the new instance and the previous instances are similar. After recording all the historic information, a new data point is categorized to use the K-NN technique based on similarity. This indicates that new information can be reliably and quickly categorized using the K-NN approach. The K-NN technique can be used for regression even though for existing approaches it is most typically applied. K-NN makes no assumptions about the basics of the data because it is not a parametric approach. As a result of saving the training sample rather than instantly learning from it, the method is also referred to as a slow learner. Instead, it executes a task while categorizing data by using the dataset. The KNN technique stores the data during the training process and classify it into a class that is similar to the new data when it is obtained.

Accuracy Achieved =83.33

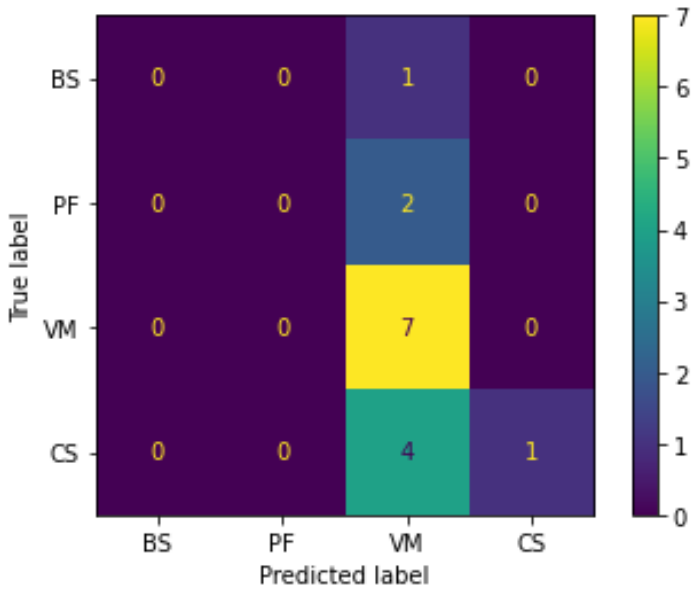

Figure 15. KNN Confusion Matrix

6.6. Logistic regression

This classification approach constructs its models from the category using a unique multiple numbers' logistic regression method with a unique estimation criterion. In a specific process, logistic regression frequently pinpoints the position of the class limit and observes that class possibilities change depending on how far away from the limit they are. This moves closer to the limits (0 and 1) increasingly faster for a huge data set. These statistical assertions set logistic regression apart from basic classifications. It is adaptable and can produce projections that are bolder and even more precise, but those exact projections could be inaccurate. Logistic regression is a technique for forecasting, much similar to Ordinary Least Squares regression. But with logistic regression, the outcome of the prediction is forked [96]. Logistic regression is the most popular technique used for the application of analysis of statistical and discrete datasets. Linear interpolation is what logistic regression is[95].

Accuracy Achieved = 79.16

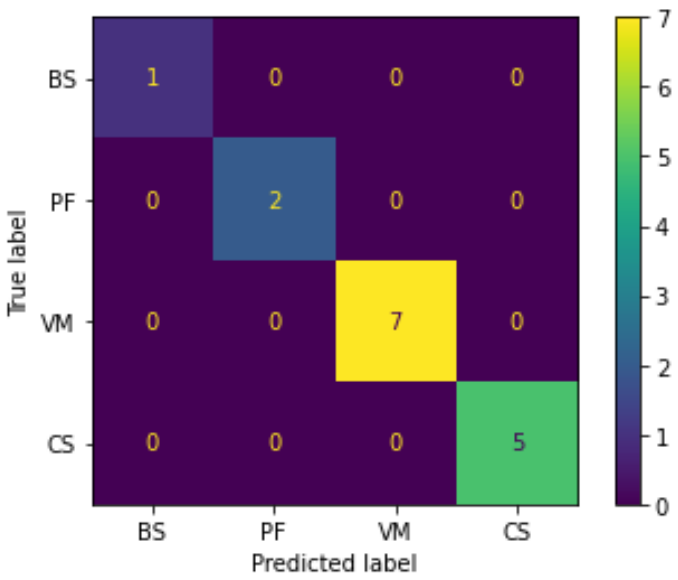

Figure 16. Logistic Regression Confusion Matrix

6.7. ANN

Computer structures called artificial neural networks (ANNs), often referred to as neural networks (NNs) or neural nets, are modelled after the neurons seen in the human brain.

An artificial neural network (ANN) is built on artificial neurons, which are a collection of linked elements or terminals that roughly simulate the neurons found in the human brain. Each link has the ability to communicate with nearby neurons, just like neurotransmitters do in the human brain. After receiving signals provided to it, a perceptron can trigger neurons that are linked to it. The "signal" at a link is a true figure, and the outcome of every neuron is determined by certain non-linear function of the combination of its inputs. Links are referred to as edges. As knowledge is acquired, the sensitivity of neurons and edges fluctuates. The density changes the signal strength of a link by boosting or lowering it. Neurons might have a limitation that must be crossed for them to deliver a signal.

Neurons commonly form layers by grouping altogether. Various layers may alter their feeds in various ways. From the initial layer to the final output layer, signals pass through the layers, maybe several times .

Accuracy Achieved = 93.75%
Test loss = 21.76%

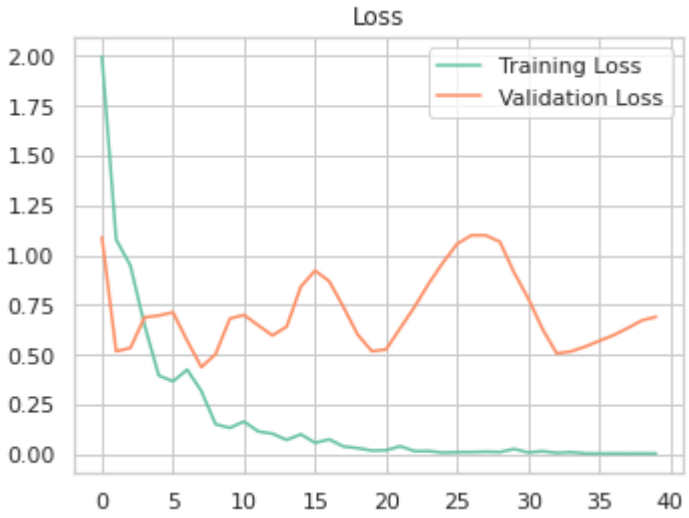

Figure 17. Graphical representation of the training loss and validation loss of data.

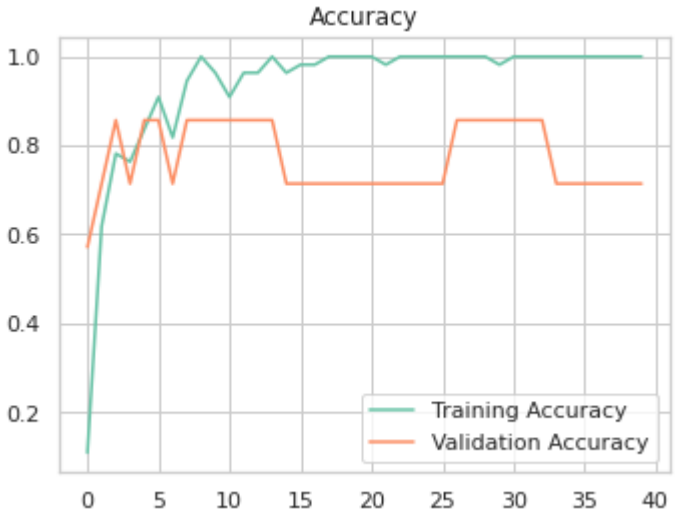

Figure 18. Graphical representation of the training set accuracy and validation set accuracy of data.

Table 5.Results of different Machine Learning Models

Model Name	Accuracy	Precision	Recall	F-Score	K-fold (k=5)
Random Forest	87.5	87.5	87.5	87.5	79.66
KNN	83.33	83.33	83.33	83.33	76.25
Decision Tree	70.83	70.8	70.8	70.8	74.33
Naïve Bayes	66.66	66.7	66.7	66.7	79.25
Logistic Regression	79.16	79.2	79.2	79.2	78.23
SVM	68.75	68.8	68.8	68.8	77.16

Table 6 .Results of Deep Learning Models

Model Name	Test Accuracy	Test Loss	Validation Accuracy	Validation Loss
Artificial Neural Network	93.75%	16.25%	98.39%	7.19%
Convolutional Neural Network	81.25%	52.21%	91.94	25.41

Chapter 7

Discussion and Future Work

7.1. Pros of the Proposed Methodology

We have done training and testing on 8 Machine Learning Models and get the highest accuracy on Artificial Neural Network that is 93.75 Accuracy. For architecture styles evaluation, analyze them, and compare them with the suggested methods we have chosen four different methods, in accordance with the linked task area. These comparison's findings are shown in Table 5 and Table 6.

Table 5 and 6 shows our survey of the various perspectives on the architectural evaluation approach. Some of them can have unique and customized cases that make it difficult to compare them to our approach. With the exception of scenario-based methods, our technique is more reusable than the others in comparison. This is because scenario-based methods can be planned to be reusable, and the architect is the only one who can make this choice. According to the evaluation description table, our method offers sequential evaluation for scalability due to the quantitative capabilities and measurement techniques based on a simulation that is simpler to scale than our method due to pre-existing simulation platforms and tools.

The suggested method is superior to both cost-based and formal methods and offers more insight into the evaluation process. We categorically cannot, however, compare our approach to the scenario-based and measuring ones. Table 5 shows that there is no need to understand simulators, formal hard procedures, or even calculate the price for various components. As a result, our approach is superior to others. This perspective also emphasizes the importance of splitting the problem into smaller sections. We cannot make a definitive assessment of our approach's completeness in comparison to scenario-based and measurement approaches because it’s fully originator and developer of the evaluation methods dependent. However, we can benefit from a variety of concerns regarding the formal and cost-based procedures.

The proposed approach has a lower cost than any of the alternatives for producing acceptable results. In fact, the expert only performs a pairwise comparison and does not even account for the expense of defining situations. It is evident that our method is more accurate than formal and measured ones in terms of precision. However, in comparison to other approaches and in its category, the hierarchical structure used in the evaluation allows for the identification and correction of flaws at each stage. This makes it the top technique for Architecture Prediction. Because it offers characteristics like modularity, step-by-step processes, input and output isolation, and checking, accountability is one of its key aspects. Despite this, it is evident that our method performs better than formal methods. No assessment can be made of the measurement-based approaches because eight quality attributes are considered in this case.

7.2. Cons of the Proposed Method

In terms of evaluating software design, each viewpoint has its own set of pros and cons that should be considered. There are some hazards associated with our suggested method for weights calculation in the different domains, such as rising prices or declining correctness (because of the involvement of human factor). In these cases, we should make use of certain supplementary techniques to eliminate these hazards or to lower the likelihood of their occurrence and detrimental effects.

The AASP related questionnaires technique may be helpful in avoiding inconsistent responses. The absence of sufficient understanding regarding the importance of each quality feature or architectural style, however, can be seen as a possible risk. As we will discuss in subsequent works (see Conclusion), we can employ a knowledge management system (quality attributes dictionary) to lower the risks. It is important to note that to avoid deceiving people and making arbitrary judgements, obtaining, and recording knowledge from architects would require additional care.

7.3. Conclusion:

One of the most important aspects in any software development process is choosing an acceptable software architecture, and meeting quality standards is one of the biggest hurdles in this field. While selecting architecture styles, it is also necessary to consider the relationships between these qualitative qualities. As a result, the style selection problem becomes a multi-criteria problem when more information is added that needs to

be studied and evaluated. Other issues in this context include the lack of quality measure evaluation standards and the unpredictability of prioritization. An effective way for analyzing architectures and choosing the best option to solve these issues is a multicriteria decision-making method. As a result, this study suggests using a Normalization technique to assess and contrast potential architectural designs. Using the quality attributes theory, this procedure uses a multi-criteria decision-making approach to address the issues brought on by information uncertainty. The suggested method provides a ranking of all possibilities along with a verifiable evaluated justification from Experts.

The task of analyzing and rating software architectures has been improved, by the application of machine learning methods based on software reengineering operations. By incorporating this into the system, the proposed method makes this strategy practical. Therefore, it produced better real-world outcomes for architects by automating architecture style prediction rather than weights assigned by the experts and surveys. The proposed method's incorporation into the development tools and application to operational tasks are both appropriate future efforts.

Description documents of Software Architectural style can also be generated using NLP in which all modules can be explicitly defined. Architecture Style can also be designed automatically using different GUI frameworks. This research is considering 9 architecture Styles, but scope can be enhanced by incorporating other Architecture Styles in the system also.

References

[1] Bosch, Jan. "Software architecture: The next step." Software architecture.Springer Berlin Heidelberg, 2004.194-199.

[2] Fairbanks, George. Just enough software architecture: a risk-driven approach. Marshall & Brainerd, 2010

[3] Lindström, Åsa, et al. "A survey on CIO concerns-do enterprise architecture frameworks support them?" Information Systems Frontiers 8.2 (2006): 81-90

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

[5] Shaw, Mary, and Paul Clements. "The golden age of software architecture." Software, IEEE 23.2 (2006): 31-39.

[6] Buschmann, Frank, Kelvin Henney, and Douglas Schimdt. Pattern-oriented Software Architecture: On Patterns and Pattern Language. Vol. 5. John wiley& sons, 2007

[7] Garlan, David. "Software architecture: a roadmap." Proceedings of the Conference on the Future of Software Engineering.ACM, 2000

[8] Gulia, S., & Choudhury, T. (2016, January). An efficient automated design to generate UML diagram from Natural Language Specifications. In 2016 6th international conference-cloud system and big data engineering (Confluence) (pp. 641-648). IEEE.

[9] Omer, O. S. D., & Mahmoud, M. M. (2021). Requirements and Design Consistency: A Bi-directional Traceability and Natural Language Processing Assisted Approach. European Journal of Engineering and Technology Research, 6(3), 120-129.

[10] Gokyer, G., Cetin, S., Sener, C., & Yondem, M. T. (2008, October). Non-functional requirements to architectural concerns: ML and NLP at crossroads. In 2008 The Third International Conference on Software Engineering Advances (pp. 400-406). IEEE.

[11] Svahnberg, M., Wohlin, C., Lundberg, L., & Mattsson, M. (2002, July). A method for understanding quality attributes in software architecture structures. In Proceedings of the 14th international conference on Software engineering and knowledge engineering (pp. 819- 826).

[12] Baseer, K. K., Reddy, A. R. M., & Bindu, C. S. A Survey of Synergistic Relationships For Designing Architecture: Scenarios, Quality Attributes, Patterns, Decisions, Reasoning Framework.

[13] Lundberg, L., Bosch, J., Häggander, D., & Bengtsson, P. O. (1999, October). Quality attributes in software architecture design. In Proceedings of the IASTED 3rd International Conference on Software Engineering and Applications (pp. 353-362). IASTED/Acta Press. Khan, Q., Qamar, U., Butt, W. H., & Rehman, S. (2017, September).

[14] Dataset designing of software architectures styles for analysis through data mining clustering algorithms. In 2017 Intelligent Systems Conference (IntelliSys) (pp. 400-405). IEEE.

[15] Khalil, M. (2013). Pattern-based methods for model-based safety-critical software architecture design: A PhD thesis proposal. Software Engineering 2013-Workshopband.

[16] Makoondlall, Y. (2020). A requirement-driven approach for modelling software architectures (Doctoral dissertation, Kingston University).

[17] Ahmed, M., Khan, S. U. R., & Alam, K. A. (2021). QAExtractor: A Quality Attributes Extraction Framework in Agile-Based Software Development. In

Proceedings of the First International Workshop on Intelligent Software Automation (pp. 15-28). Springer, Singapore.

[18] Klein MH, Kazman R, Bass L, Carriere J, Barbacci M, Lipson H (1999) Attribute-based architecture styles. In: Software architecture. Springer Berlin, pp 225–243

[19] Svahnberg M (2003) Supporting software architecture evolution Doctoral dissertation, Blekinge Institute of Technology

[20] Abowd G, Bass L, Clements P, Kazman R, Northrop L (1997) Recommended best industrial practice for software architecture evaluation (No. CMU/SEI-96-TR-025). Carnegie-Mellon University, Pittsburgh PA, Software Engineering Institute

[21] Maranzano J (1993) Best current practices: software architecture validation. AT&T Report

[22] Schmidt D, Stal M, Rohnert H, Buschmann F (1996) Pattern-oriented software architecture, volume 1: a system of patterns

[23] Kazman R, Bass L, Abowd G, Webb M (1994) SAAM: a method for analyzing the properties of software architectures. In: 16th international conference on software engineering, 1994. Proceedings. ICSE-16. IEEE, pp 81–90

[24] Kazman R, Klein M, Clements P (2002) ATAM: method for architecture evaluation: ATAM—architecture trade-of analysis method report

[25] Paul C, Kazman R, Klein M (2002) Evaluating software architectures: methods and case studies. AddisonÇęWesley, Boston, MA

[26] Nord RL, Barbacci MR, Clements P, Kazman R, Klein M (2003) Integrating the Architecture Tradeof Analysis Method (ATAM) with the cost beneft analysis method (CBAM). Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst

[27] Bengtsson P (2002) Architecture-level modifability analysis. Doctoral dissertation, Free University of Amsterdam

[28] Lassing N (2002) Architecture-level modifability analysis. Doctoral dissertation, Free University of Amsterdam

[29] Thomas J (2002) Architecture assessment of information-system families. Doctoral dissertation, Eindhoven University of Technology

[30]] Medvidovic N, Taylor RN (2000) A classifcation and comparison framework for software architecture description languages. IEEE Trans Softw Eng 26(1):70–93

[31] Garlan D, Allen R, Ockerbloom J (1994) Exploiting style in architectural design environments. ACM SIGSOFT Softw Eng Notes 19(5):175–188

[32] Medvidovic N, Oreizy P, Robbins JE, Taylor RN (1996) Using object-oriented typing to support architectural design in the C2 style. ACM SIGSOFT Softw Eng Notes 21(6):24–32

[33] Magee J, Dulay N, Eisenbach S, Kramer J (1995) Specifying distributed software architectures. In: Software engineering—ESEC'95, pp 137–153

[34] Bengtsson P (2002) Architecture-level modifability analysis. Doctoral dissertation, Free University of Amsterdam

[35] Luckham DC, Kenney JJ, Augustin LM, Vera J, Bryan D, Mann W (1995) Specifcation and analysis of system architecture using Rapide. IEEE Trans Softw Eng 21(4):336–354

[36] Shaw M, DeLine R, Klein DV, Ross TL, Young DM, Zelesnik G (1995) Abstractions for software architecture and tools to support them. IEEE Trans Softw Eng 21(4):314–335

[37] Garlan D, Monroe R, Wile D (2010) ACME: an architecture description interchange language. In: CASCON frst decade high impact papers. IBM Corp., pp 159–173

[38] Moaven S, Habibi J, Ahmadi H, Kamandi A (2008) A decision support system for software architecture-style selection. In: Sixth international conference on software engineering research, management and applications, 2008. SERA'08. IEEE, pp 213–220

[39] Moaven S, Habibi J, Ahmadi H, Kamandi A (2008) A fuzzy model for solving architecture styles selection multi-criteria problem. In: Second UKSIM European symposium on computer modeling and simulation, 2008. EMS'08. IEEE, pp 388–393

[40] Moaven S, Kamandi A, Habibi J, Ahmadi H (2009) Toward a framework for evaluating heterogeneous architecture styles. In: First Asian conference on intelligent information and database systems, 2009. ACIIDS 2009. IEEE, pp 155–160

[41] Babu K, Rajulu PG, Reddy AR, Kumari AN (2010) Selection of architecture styles using analytic network process for the optimization of software architecture. arXiv preprint arXiv:1005.4271

[42] Galster M, Eberlein A, Moussavi M (2010) Systematic selection of software architecture styles. IET Softw 4(5):349–360

[43] Zaki MZ, Jawawi DN, Hamdan NM, Halim SA, Mamat R, Mahat FS, Omar NA (2013) Multi-criteria architecture style selection for precision farming software product lines using fuzzy AHP. Int J Adv Soft Comput Appl 5(3):85

[44] Dwivedi AK, Rath SK (2014) Selecting and formalizing an architectural style. In: 2014 seventh international conference on contemporary computing (IC3). IEEE, pp 364–369

[45] Tahmasebipour S, Babamir SM (2014) Ranking of common architectural styles based on availability, security and performance quality attributes. J Comput Secur 1(2):83–93

[46] Tan C, Chen X (2010) Intuitionistic fuzzy Choquet integral operator for multi-criteria decision making. Expert Syst Appl 37(1):149–157

[47] Babu KD, Govindaraju P, Reddy AR (2011) ANP-GP approach for selection of software architecture styles. Int J Softw Eng 1(5):91–104

[48] Dhaya C, Zayaraz G (2012) Fuzzy based quantitative evaluation of architectures using architectural knowledge. Int J Adv Sci Technol 49:137–154

[49] Saaty TL (1977) A scaling method for priorities in hierarchical structures. J Math Psychol 15(3):234–281

[50] Harker PT, Vargas LG (1987) The theory of ratio scale estimation: Saaty's analytic hierarchy process. Manag Sci 33(11):1383–1403

[51] Abrahão S, Insfran E (2017) Evaluating Software Architecture Evaluation Methods: An Internal Replication. In: Proceedings of the 21st international conference on evaluation and assessment in software engineering. ACM, pp 144–153

[52] Mahdavi-Hezavehi S, Durelli VH, Weyns D, Avgeriou P (2017) A systematic literature review on methods that handle multiple quality attributes in architecture-based self-adaptive systems. Inf Softw Technol 90:1–26

[53] Montagud S, Abrahão S, Insfran E (2012) A systematic review of quality attributes and measures for software product lines. Softw Quality J 20(3–4):425–486

[54] Moaven S, Habibi J, Alidoosti R, Mosaed AP (2015) Towards a knowledge based approach to style driven architecture design. Procedia Comput Sci 62:236–244

[55] Dasanayake, S., Markkula, J., Aaramaa, S., & Oivo, M. (2015, September). Software architecture decision-making practices and challenges: an industrial case study. In: 2015 24th Australasian software engineering conference (ASWEC). IEEE, pp. 88–97

[56] Hayes-Roth, Barbara, et al. "A domain-specific software architecture for adaptive intelligent systems." Software Engineering, IEEE Transactions on 21.4 (1995): 288-301.

[57] Dong, Jing, Shanguo Chen, and Jun-Jang Jeng. "Event-based blackboard architecture for multi-agent systems." Information Technology: Coding and Computing, 2005. ITCC 2005.International Conference on.Vol. 2.IEEE, 2005.

[58] Kekic, Miodrag M., Grace N. Lu, and Eloise H. Carlton. "Client-server computer network management architecture." U.S. Patent No.6,664,978. 16 Dec. 2003.

[59] http://www.cs.montana.edu/~halla/csci466/lectures/lec2.html

[60] Davis, Keir, John W. Turner, and Nathan Yocom. "Client-Server Architecture."The Definitive Guide to Linux Network Programming.Apress, 2004. 99-135. [14] http://java.boot.by/scea5-guide/ch02s02.html

[61] Heineman, George T., and William T. Council. "Component-based software engineering: putting the pieces together." (2001).

[62] Mahmood, Sajjad, Richard Lai, and Yong Soo Kim. "Survey of component-based software development." Software, IET 1.2 (2007): 57-66.

[63] Garlan, David, Robert T. Monroe, and David Wile. "Acme: Architectural description of component-based systems." Foundations of component-based systems 68 (2000): 47-68.

[64] Herzum, Peter, and Oliver Sims. Business Components Factory: A Comprehensive Overview of Component-Based Development for the Enterprise. John Wiley & Sons, Inc., 2000.

[65] http://docs.apiwatch.org/en/apiwatch-0.1/user/basic-concepts/

[66] Bruns, Ralf, ed. Event-Driven Architecture. Springer Berlin Heidelberg, 2010.

[67] Chandy, K. Mani. "Event Driven Architecture."Encyclopedia of Database Systems.Springer US, 2009.1040-1044.

[68] http://www.ibm.com/developerworks/library/ws-eventprocessing/

[69] Mehta, Nikunj R., and NenadMedvidovic. "Composing architectural styles from architectural primitives."ACM SIGSOFT Software Engineering Notes.Vol. 28.No. 5.ACM, 2003.

[70] Rapanotti, Lucia, et al. "Architecture-driven problem decomposition." Requirements Engineering Conference, 2004.Proceedings.12th IEEE International.IEEE, 2004.

[71] msdn.microsoft.com/en-us/library/ee658109.aspx

[72] Gorton I (2006) Essential software architecture. Springer, Berlin

[73] ISO/IEC, ISO/IEC CD 25010.3 (2009) Systems, software engineering - Software product Quality Requirements and Evaluation (SQuaRE) Software product quality, and system quality in use models, ISO

[74] https://opennlp.apache.org/

[75] *Overview CoreNLP*. Available at: https://stanfordnlp.github.io/CoreNLP/ (Accessed: November 6, 2022).

[76] *NLTK*. Available at: https://www.nltk.org/ (Accessed: November 6, 2022).

[77] *Index.html GATE*. Available at: https://gate.ac.uk/ (Accessed: November 6, 2022).

[78] Spacy · industrial-strength natural language processing in python · Industrial-strength Natural Language Processing in Python. Available at: https://spacy.io/ (Accessed: November 6, 2022).

[79] Collab-Uniba (no date) *Collab-uniba/emotion_and_polarity_so: An emotion classifier of text containing technical content from the se domain, GitHub*. Available at: https://github.com/collab-uniba/Emotion_and_Polarity_SO (Accessed: November 6, 2022).

[80] *Machine learning for language toolkit Mallet*. Available at: http://mallet.cs.umass.edu/ (Accessed: November 6, 2022).

[81] Team, K. *Simple. flexible. powerful., Keras*. Available at: https://keras.io/ (Accessed: November 6, 2022).

[82] Gulia, S., & Choudhury, T. (2016, January). An efficient automated design to generate UML diagram from Natural Language Specifications. In 2016 6th international conference-cloud system and big data engineering (Confluence) (pp. 641-648). IEEE

[83] J. Hirschberg and C. D. Manning. 2015. Advances in natural language processing. Science 349, 6245, (2015), 261–266.

[84] E. D. Liddy. 2001. Natural language processing. In Encyclopedia of Library and Information Science, 2nd ed. Marcel Decker, New York, NY.

[85] M. Galster, A. Eberlein, and M. Moussavi, “Systematic selection of software architecture styles,” Software, IET, vol. 4, no. 5, pp. 349–360, 2010.

[86] Chang DY (1996) Applications of the extent analysis method on fuzzy AHP. Eur J Oper Res 95(3):649–655

[87] Chang CW, Wu CR, Lin HL (2009) Applying fuzzy hierarchy multiple attributes to construct an expert decision making process. Expert Syst Appl 36(4):7363–7368

[88] Van Laarhoven PJM, Pedrycz W (1983) A fuzzy extension of Saaty's priority theory. Fuzzy Sets Syst 11(1–3):229–241

[89] Saaty TL (1990) How to make a decision: the analytic hierarchy process. Euro J Oper Res 48(1):9–26

[90] Saaty TL (1980) The analytic hierarchy process, paperback edition. RWS Publications, Pittsburgh (First appeared)

[91] Domingos, P. & Pazzani, M. (1997). On the optimality of the simple Bayesian classifier under zero-one loss. Machine Learning Volume 29, pp. 103–130 Copyright © 1997 Kluwer Academic Publishers. Manufactured in The Netherlands. Available at University of Trento website: http://disi.unitn.it/~p2p/RelatedWork/Matching/domingos9 7optimality.pdf

[92] Good, I.J. (1951). Probability and the Weighing of Evidence, Philosophy Volume 26, Issue 97, 1951. Published by Charles Griffin and Company, London 1950.Copyright © The Royal Institute of Philosophy 1951,pp. 163-164.doi: https://doi.org/10.1017/S0031819100026863. Availableat Royal Institute of Philosophy website: https://www.cambridge.org/core/journals/philosophy/article /probability-and-the-weighing-of-evidence-by-goodi-jlondon-charles-griffin-and-company-1950-pp-viii-119- price-16s/7D911224F3713FDCFD1451BBB2982442

[93] Hormozi, H., Hormozi, E. & Nohooji, H. R. (2012). The Classification of the Applicable Machine Learning Methods in Robot Manipulators. International Journal of Machine Learning and Computing (IJMLC), Vol. 2, No. 5, 2012 doi: 10.7763/IJMLC.2012.V2.189pp. 560 – 563. Available at IJMLC website: http://www.ijmlc.org/papers/189-C00244-001.pdf

[94] Kotsiantis, S. B. (2007). Supervised Machine Learning: A Review of Classification Techniques. Informatica 31 (2007). Pp. 249 – 268. Retrieved from IJS website: http://wen.ijs.si/ojs2.4.3/index.php/informatica/article/download/148/140.

[95] Logistic Regression pp. 223 – 237. Available at: https://www.stat.cmu.edu/~cshalizi/uADA/12/lectures/ch12 .pdf

[96] Newsom, I. (2015). Data Analysis II: Logistic Regression. Available at: http://web.pdx.edu/~newsomj/da2/ho_logistic.pdf

[97] Nilsson, N.J. (1965). Learning machines. New York: McGraw-Hill.Published in: Journal of IEEE Transactions on Information Theory Volume 12 Issue 3, 1966. doi: 10.1109/TIT.1966.1053912 pp. 407 – 407. Available at ACM digital library website: http://dl.acm.org/citation.cfm?id=2267404

[98] T. Hastie, R. Tibshirani, J. H. Friedman (2001) ― The elements of statistical learning,‖ Data mining, inference, and prediction, 2001, New York: Springer Verlag.

Vapnik, V. N. (1995). The Nature of Statistical Learning Theory. (2nd ed.). Springer Verlag. Pp. 1 – 20. Retrieved from website: https://www.andrew.cmu.edu/user/kk3n/simplicity/vapnik2 000.pdf