



جامعة طيبة

كلية علوم وهندسة الحاسب الآلي

(قسم الطلاب)

KINGDOM OF SAUDI ARABIA

Ministry of Education

Taibah University

College of Computer Science and

Engineering

(Male Section)

**AI-Powered Requirements Analysis Modeling**

**Graduation Project 2**

**by**

Ali Talal Al-Ahmadi 4100379

Mohammed Hadi Al-Harbi 4101704

Rami Ramadan Al-Mohammadi 4101758

**A project submitted in partial fulfilment of the requirements for the degree of Bachelor of Science Computer Science**

**Supervised by**

Dr. Mohammad M. Alsuraihi

1st Semester - Academic Year 1445 (2023/2024)

# Abstract

The project may face issues with excessive resource consumption, such as time or costs, and frequent changes in requirements can lead to project delays and increased expenses. Additionally, sometimes the techniques or tools used in the analysis and design process are ineffective or outdated. Software maintenance constitutes a pivotal stage within the software development lifecycle, encompassing a substantial portion, varying from 40% to 80% of the total expenses associated with software development. It's worth highlighting that a notable 60% of the overall maintenance expenditure is dedicated to the improvement of existing software functionalities. Consequently, it becomes imperative to meticulously prepare appropriate software documentation at each developmental phase in order to alleviate the financial burdens of maintenance. Using artificial intelligence (AI) to analyze requirements and generate drawings or models can be an effective solution for saving time and costs in the design and analysis processes. The crux of reducing maintenance costs lies in enhancing one's grasp of the software system, as understanding a software system accounts for roughly 50% of the time spent in the maintenance phase.

**Keywords** NLP; ML; AI-Powered; Requirements; Analysis; UML

# Acknowledgement

Alhamdulillah. First and foremost, all praise is due to Allah SWT the worthy of all the praises and compliments for giving us the strength and ability to complete this project. Next, we express gratitude to our parents and families. We would like to extend our thanks to our project supervisor, Dr. Mohammad M. Alsuraihi, who guided us in executing this project, providing invaluable advice, assisting us in challenging times, and significantly contributing to the project's completion.

We also want to express our appreciation to all the professors at Taibah University, College of Engineering and Computer Science, who have been part of our academic journey at the university.

Finally, and not least, we would like to thank those who worked on this project: Ali Talal Al-Ahmadi, Mohammed Hadi Al-Harbi, and Rami Ramadan Al-Mahmoudi, for their contributions to this project. I appreciate their hard work and dedication.

**Contents**

[Abstract ii](#_Toc166090617)

[Acknowledgement iii](#_Toc166090618)

[List of Figures vii](#_Toc166090619)

[List of Tables viii](#_Toc166090620)

[List of Abbreviations ix](#_Toc166090621)

[1 Chapter 1: Introduction 1](#_Toc166090622)

[1.1 Introduction 1](#_Toc166090623)

[1.2 Problem Definition 2](#_Toc166090624)

[1.3 Project Aim and Objectives 2](#_Toc166090625)

[1.4 Project Methodology 3](#_Toc166090626)

[1.5 Project Timeline 4](#_Toc166090627)

[1.6 Document Organization 5](#_Toc166090628)

[1.7 Summary 5](#_Toc166090629)

[2 Chapter 2: Literature Review 7](#_Toc166090630)

[2.1 Introduction 7](#_Toc166090631)

[2.2 Research Methodology 7](#_Toc166090632)

[2.3 AI 8](#_Toc166090633)

[2.4 Machine Learning 9](#_Toc166090634)

[2.5 NLP 9](#_Toc166090635)

[2.5.1 Normalization 10](#_Toc166090636)

[2.5.2 Vectorization of Text 10](#_Toc166090637)

[2.6 System Development for AI 11](#_Toc166090638)

[2.6.1 NLP application Development 11](#_Toc166090639)

[2.6.1 ML application Development 12](#_Toc166090640)

[2.7 Similar Tools 14](#_Toc166090641)

[2.7.1 Software Requirements Classification Using Machine Learning Algorithms 14](#_Toc166090642)

[2.7.2 From user requirements to UML class diagram 18](#_Toc166090643)

[2.7.3 AI Based UML Diagrams Generator 22](#_Toc166090644)

[2.8 Summary 30](#_Toc166090645)

[3 Chapter 3: System Analysis 31](#_Toc166090646)

[3.1 Introduction 31](#_Toc166090647)

[3.2 Methodology 31](#_Toc166090648)

[3.2.1 SDLC 31](#_Toc166090649)

[3.2.2 Analysis 32](#_Toc166090650)

[3.3 Analysis of Existing Systems 33](#_Toc166090651)

[3.4 Requirements Elicitation 34](#_Toc166090652)

[3.4.1 System Requirements 34](#_Toc166090653)

[3.4.2 Functional Requirements 35](#_Toc166090654)

[3.4.3 Non-Functional Requirements 35](#_Toc166090655)

[3.5 System Scenario 36](#_Toc166090656)

[3.6 Requirements Specification 37](#_Toc166090657)

[3.6.1 Usecase 1: 37](#_Toc166090658)

[3.6.2 Usecase 2: 37](#_Toc166090659)

[3.6.3 Usecase 3: 38](#_Toc166090660)

[3.6.4 Usecase 5: 38](#_Toc166090661)

[3.6.5 Usecase 6: 38](#_Toc166090662)

[3.7 Requirements Modeling 39](#_Toc166090663)

[3.7.1 usecase 39](#_Toc166090664)

[3.7.2 Class Diagram 40](#_Toc166090665)

[3.8 Summary 41](#_Toc166090666)

[4 Chapter 4: System Design 42](#_Toc166090667)

[4.1 Introduction 42](#_Toc166090668)

[4.2 Design Methodology 42](#_Toc166090669)

[4.3 Architectural Design 43](#_Toc166090670)

[4.4 Component Design 44](#_Toc166090671)

[4.4.1 First main component: document analysis 45](#_Toc166090672)

[4.4.2 First main component: Generate Diagram: 46](#_Toc166090673)

[4.5 Data Modeling Design 47](#_Toc166090674)

[4.6 User Interface Design 48](#_Toc166090675)

[4.7 Summary 50](#_Toc166090676)

[5 Chapter 5: Implementation and Testing 51](#_Toc166090677)

[5.1 Methodology 51](#_Toc166090678)

[5.2 Implementation 52](#_Toc166090679)

[5.3 Testing 59](#_Toc166090680)

[5.3.1 System testing 59](#_Toc166090681)

[5.3.2 Functional testing 64](#_Toc166090682)

[6 Chapter 6: Conclusion and Future Work 65](#_Toc166090683)

[6.1 Conclusion 65](#_Toc166090684)

[6.2 Goals Achieved 66](#_Toc166090685)

[6.3 Lessons Learnt 66](#_Toc166090686)

[6.4 Limitations and Future Work 67](#_Toc166090687)

[6.4.1 Limitations 67](#_Toc166090688)

[6.4.1 Future work 67](#_Toc166090689)

[7 References 68](#_Toc166090690)

# List of Figures

[Figure 1: Project Methodology 4](#_Toc166090500)

[Figure 2: Phases of requirements classification pipeline. 14](#_Toc166090501)

[Figure 3: Number of requirements per label. 15](#_Toc166090502)

[Figure 4: Top 10 most important features. 16](#_Toc166090503)

[Figure 5: Results of binary classification (FR and NFR). 17](#_Toc166090504)

[Figure 6: Results of NFR classification. 18](#_Toc166090505)

[Figure 7: Results of FR and NFR classification (12 granularities). 18](#_Toc166090506)

[Figure 8 DC-Builder System architecture. 20](#_Toc166090507)

[Figure 9 Proposed Solution Design 22](#_Toc166090508)

[Figure 10 Process flow with NLP And ML Model 23](#_Toc166090509)

[Figure 11 Sample dataset 25](#_Toc166090510)

[*Figure 12 confusion matrix for the case of binary classification* 29](#_Toc166090511)

[Figure 13: waterfall for project. 32](#_Toc166090512)

[Figure 14: Description of stages of analysis 33](#_Toc166090513)

[Figure 15: user use-case diagram 39](#_Toc166090514)

[Figure 16: class diagram for our system. 40](#_Toc166090515)

[Figure 17:Design methodolgy 42](#_Toc166090516)

[Figure 18: Architectural Design 44](#_Toc166090517)

[Figure 19:Component Diagram 45](#_Toc166090518)

[Figure 20: Home Page 48](#_Toc166090519)

[Figure 21: Input Page 48](#_Toc166090520)

[Figure 22: Diagram Page 49](#_Toc166090521)

[Figure 23: Implementation and Testing Methodology 51](#_Toc166090522)

[Figure 24: Showcases the findActor(doc) method in our system. 52](#_Toc166090523)

[Figure 25: Showcases the findUsecase(doc, actors) method in our system. 53](#_Toc166090524)

[Figure 26: Showcases the findUsecaseRelationship(doc, actors) method in our system. 55](#_Toc166090525)

[Figure 27: Showcases the findClass(doc) method in our system. 56](#_Toc166090526)

[Figure 28: Showcases the findMethod(doc) method in our system. 57](#_Toc166090527)

[Figure 29: Showcases the findAttributes(doc) method in our system. 58](#_Toc166090528)

# List of Tables

[Table 1: Project plan 4](#_Toc166090529)

[Table 3: Related Work Comparison 34](#_Toc166090530)

[Table 3: use-case (analysis documents) 37](#_Toc166090531)

[Table 4: use-case (Enter requirements Document) 37](#_Toc166090532)

[Table 5: use-case (Create analysis model) 38](#_Toc166090533)

[Table 6: use-case (Draw use-case model) 38](#_Toc166090534)

[Table 7: use-case (Draw class model) 38](#_Toc166090535)

[Table 8: Actual data usecase elements 60](#_Toc166090536)

[Table 9: Our system output usecase 61](#_Toc166090537)

[Table 10: Evaluation metric result for usecase elements 61](#_Toc166090538)

[Table 11: Actual data class diagram elements 62](#_Toc166090539)

[Table 12: Our system output class diagram elements 63](#_Toc166090540)

[Table 13: Evaluation metric result for usecase elements 63](#_Toc166090541)

[Table 14: Our system Functional testing 64](#_Toc166090542)

# List of Abbreviations

UML Unified Modeling Language

ML Machine Learning

NLP Natural Language Processing

AI Artificial Intelligence

UI User Interface

RNN Recurrent Neural Network

SDLC Software Development Life Cycle

NNS Neural Network Systems

NER Named Entity Recognition

# Chapter 1: Introduction

## Introduction

Nowadays, the field of software development is using the capabilities of artificial intelligence to improve the phases of this field, including prediction and decision-making assistance. Artificial intelligence has become instrumental in guaranteeing the high quality of software development phases. Additionally, it has the capacity to significantly reduce the time needed for software development, leading to increased overall productivity. [1]

Software development involves creating and maintaining computer programs using various tools and methodologies to produce high-quality software efficiently this field involves using various programming languages, implementing different development methodologies like Agile and Waterfall

This project focuses on using Machine Learning for software development, specifically for converting software requirements and specifications into analysis models. The project was split into two phases. The first phase aimed at completing the analysis and part of the design for our proposed system. The work on that phase was tested and a number of notes were taken. These are the most important notes:

* Give more focus on related work and systems that employ Machine Learning for software analysis and modeling.
* Writing scenarios on applying Machine Learning and Natural Language Processing algorithms on chosen data samples at the analysis phase.
* Revise the analysis models (function and object model).
* Enhance the architecture design.

It was also recommended to use the Python programming language and libraries (e.g., scikit-learn library for predictive data analysis

This chapter highlights the goal and objectives of this project, outlines our methodology to achieve these objectives, gives the plan for the project’s tasks, provides an overview of the subsequent chapters in this report, and finalizes with an overall summary of the chapter.

## Problem Definition

Software maintenance constitutes a pivotal stage within the software development lifecycle, encompassing a substantial portion, varying from 40% to 80% of the total expenses associated with software development. It's worth highlighting that a notable 60% of the overall maintenance expenditure is dedicated to the improvement of existing software functionalities. Consequently, it becomes imperative to meticulously prepare appropriate software documentation at each developmental phase in order to alleviate the financial burdens of maintenance. The crux of reducing maintenance costs lies in enhancing one's grasp of the software system, as understanding a software system accounts for roughly 50% of the time spent in the maintenance phase. To facilitate this understanding, various modeling languages have surfaced, enabling graphical representations that substantially contribute to an enhanced comprehension of software systems. [2]

The project may face issues with excessive resource consumption, such as time or costs, and frequent changes in requirements can lead to project delays and increased expenses.

Using artificial intelligence (AI) to analyze requirements and generate drawings or models can be an effective solution for saving time and costs in the design and analysis processes.

## Project Aim and Objectives

The objective of the project is to deepen our understanding of the related works in terms of design and code to utilize the algorithms and technologies used in the related works, improve the system analysis process, complete the design phase with a focus on the data model and components, and program and test the system. All of this is aimed at accelerating and facilitating the work of the system modeling analysts and saving their time.

To achieve this goal, we must achieve the following objectives:

1. To review and delve deeper into related work in terms of design and code.
2. Improve the system analysis process.
3. Complete the design phase with a focus on the data model and components.
4. Program and test the system.
5. Conclude with the lessons learned and the knowledge and experience gained from working on this project.

## Project Methodology

Reviewing and deepening past works in terms of design and code is a key step in the research phase of our project. By examining current research and projects in our project field, we can gain valuable insights into the latest developments, identify potential challenges, and build on the knowledge and results of others.

Improving the analysis phase is important for the development of our project. During this phase, we thoroughly examine project requirements, analyze current systems and processes. It provides the basis for all subsequent project activities, ensuring that we have a clear understanding of what to achieve and how to achieve it.

After completing the Analysis phase, we move on to the preliminary Design tasks. This phase involves translating the gathered requirements and insights into a comprehensive system design. We will define the system's architecture, data flows, and functionality.

After completing the design phase, we move to the implementation phase and validate the system.

Concluding the project with a comprehensive review of lessons learned, knowledge gained, and experiences accumulated is essential for continuous improvement. This reflection allows us to identify areas of success and areas for improvement in project management, technical implementation, and collaboration. By documenting our insights and best practices, we can enhance our future projects and contribute to the growth of our team's expertise.



Figure 1: Project Methodology

## Project Timeline

This section presents the timeline plan versions of our project as the following:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **May** | | **April** | | | | **March** | | | | **February** | | | | **January** | **Months**  **(2024)** |
| **2** | **1** | **4** | **3** | **2** | **1** | **4** | **3** | **2** | **1** | **4** | **3** | **2** | **1** | **4** | **Weeks** |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | Plan | **Tasks** |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | Analysis and design of the system |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | Complete implementation and validate phase |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | Documentation |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | Presentation |

Table 1: Project plan

1. **Plan (3 weeks):**

"We will define the main objectives of the project and review all the notes from previous discussions. Additionally, we will review the existing systems to understand what languages were used in developing the previous systems and how they function.".

1. **Analysis and design of the system (2 weeks):**

We will work on improving the system analysis process to ensure we get the most accurate results. We will enhance the system design and draw data models and components.

1. **Complete implementation and validate phase (6 weeks):**

We will program the system and test it to ensure it works as expected.

1. **Documentation:**

This task spans throughout the project's duration as it involves continuous documentation of the work.

1. **Project Presentation:**

The project should be presented during this time to showcase the results.

## Document Organization

* **Chapter 1.** In this chapter we discussed the plan and objectives and identified the problem that the project solves.
* **Chapter 2.** In this chapter, we reviewed existing systems.
* **Chapter 3.** Our focus in this chapter was on identifying functional and non-functional requirements using UML diagrams, as well as identifying user requirements and search methodologies.
* **Chapter 4.** In this chapter, we designed the system, identified the components of the system, modeled the data, and designed the user interfaces.
* **Chapter 5.** In this chapter, we programmed the system and conducted a series of tests to ensure that the system is working well.
* **Chapter 6.** A presentation of the project's conclusion was given, along with a proposal for new work to improve the current work, and a statement of whether the original objectives of the project had been met.

## Summary

In this chapter, we documented the feedback from the project's first graduation thesis and outlined the project's goals. We also defined a plan to address the feedback and execute the project modifications.

In the second chapter, we will delve into similar previous works, gathering and discussing all relevant information.

# Chapter 2: Literature Review

## Introduction

In this chapter, we meticulously explore the current scholarly landscape within our field of study. Our primary objective is to conduct a precise examination of the existing literature and research pertaining to our subject matter. We will study and analyze the methodologies and techniques employed in this context.

Furthermore, we will undertake a comprehensive review and comparative analysis of systems and research akin to our project. Through these reviews and comparisons, we aim to extract valuable insights that contribute to understanding the historical and contemporary developments in our field, and elucidate the patterns, trends, and innovative approaches that have shaped the discourse and research.

By conducting a meticulous evaluation of prior works and engaging in a profound comparison with our research approach, we aim to direct our efforts towards delivering novel contributions and deepening our understanding of our study's domain.

## Research Methodology

Our data collection process involves using the Google Scholar search engine to find diverse scientific literature sources, such as research papers, conference papers, articles, and e-books that can assist us in understanding similar studies and identifying their strengths, weaknesses, similarities, and differences. For citation and referencing, we use digital libraries from IEEE, ACM, the university library, trusted websites (e.g., .org, .edu), and the Saudi Digital Library. Additionally, we use ChatGPT to assist us in translating and comprehending texts.

We will determine the Software Development Life Cycle (SDLC) model, whether we will use the Waterfall or Agile model, in Chapter 3 after researching topics related to our project and previous systems.

1. **IEEE** [3]**:** The IEEE style is a numerical style, where citations are numbered according to their appearance order. This number guides the reader to a complete reference in the reference list at the end of the work. The citation number should be placed within square brackets on the same line as the text, before any punctuation, with a space before the square brackets, such as: [x]. Once cited from a source, the same number is reused for all subsequent citations from the same source. [4]
2. **ACM Library Online** [5]**:** The ACM Digital Library is a research, discovery, and networking platform that offers full-text access to all ACM publications, including journals, conference proceedings, technical magazines, newsletters, and books. It also maintains a carefully curated collection of full-text publications from a limited number of publishers. The ACM Guide to Computer Literature is a searchable database dedicated solely to computing literature, encompassing a complex network of relationships among authors, works, institutions, and specialized communities.

## AI

Artificial intelligence (AI) is the branch of computer science concerned with making computers behave like humans (Computers with the ability to mimic or duplicate the functions of the human brain). [6]

Artificial intelligence systems encompass individuals, processes, hardware, software, data, and the essential knowledge required to develop computer and machine systems that exhibit intelligent characteristics. [6]

**Techniques used in AI:**

1. **Knowledge Representation**: It is the science of translating real-world knowledge into a form that can be used by computers.
2. **Search:** A technique for selecting the best solution from all possible solutions.
3. **Automated Reasoning:** The process of achieving a specific goal based on prior knowledge.
4. **Planning:** The ability to make a good sequence of actions to achieve our objectives.

## Machine Learning

Machine science is a branch of artificial intelligence that aims to facilitate human life by replicating human behaviors. It is used to train the machine to train itself and to teach the machine to deal with data accurately. One of its uses is to predict future results based on known or previous data. Some of its types include supervised learning, unsupervised learning, and neural networks. [7] [8]

Some machine learning models that we might use include: "There are several methods used in machine learning, including **K-NN (K-Nearest Neighbor)**, which classifies new data based on the distance between it and the items in the database, then identifies the closest k items and calculates their average for regression problems or obtains the mode for classification. Then there’s the **Support Vector Machine (SVM)**, which creates a linear boundary between two classes to minimize the likelihood of misclassifying new instances. This boundary is established with the maximum margin separating the two classes. **Logistic Regression (LR)** uses an independent variable to predict a dependent variable and can be multiclass if the dependent variable has more than two classes, and binary if it has two classes. Finally, **Multinomial Naive Bayes (MNB)** estimates the conditional probabilities for a specific class based on the input data. It assumes that input features are independent and is specifically used for classifying documents and texts.

## NLP

Natural Language Processing (NLP) is a branch of artificial intelligence that involves processing and analyzing text data and includes machine learning to understand and interact with human language. [9]

**How does a computer understand languages?**

Computers are emotionless machines, so we must convert natural languages into numbers. This numerical transformation allows the computer to perform mathematical operations on language data, enabling it to comprehend human language. [9]

### Normalization

Text normalization is the process of preparing and organizing textual data for use in Natural Language Processing (NLP) and analytics systems. It involves various steps, including tokenization, where the text is divided into individual tokens like sentences or words. Tokenization is just one aspect of text normalization. Other techniques in this process include case conversion, fixing spelling errors, removing irrelevant words like articles and pronouns, as well as stemming and lemmatization. The goal is to clean and standardize the text, making it suitable for input into NLP and analytics applications. [10]

### Vectorization of Text

Do machine learning algorithms expect input as a binary matrix, where rows represent instances and columns represent features. We need to convert our examples and documents into vector representations to implement machine learning.

Some examples of Text Vectorization, which involves analyzing text, are:

1. **Bag of Words (BoW):**

The Bag of Words (BoW) model is a technique for feature extraction from text, and it is one of the simplest and most powerful methods. This model transforms textual documents into vectors, where each document becomes a vector representing the frequency of each unique word in the document's vector space. Therefore, the document "J" is expressed by the vector. [10]

xi,j It refers to the weight of feature **i**, calculated from the frequency of term **i** in document **j**, where **n** indicates the number of terms in the requirements dictionary.

1. **TF-IDF:**

It is a model that combines two measures. The first is the initial frequency value of a specific term in a document, and the second is the inverse document frequency for each term. The inverse document frequency can be mathematically represented by the following formula:

When combining the two measures, TF-IDF can be mathematically represented by the following formula: [10]

Where **tf** represents the term frequency, **idf** is the inverse document frequency, **i** is for the term, and **j** is for the document.

1. **CHI2:**

It is a common statistical test that measures the deviation from the expected distribution if one assumes that the occurrence of the feature is actually independent of the class value. [10] It assesses the independence between the term t and the class c and is determined by the formula:

where **N** is the total number of documents, **A** is the number of times **t** and **c** co-occur, **B** is the number of times the **t** occurs without **c**, **C** is the number of times c occurs without t and **D** is the number of times neither **c** nor **t** occurs.

## System Development for AI

In this section, we will discuss the system development for Artificial Intelligence applications and the stages followed in the development of Natural Language Processing (NLP) and Machine Learning (ML) applications, along with the languages used in development.

### NLP application Development

Developing Natural Language Processing (NLP) applications is crucial in the field of artificial intelligence. This work involves several sequential steps to ensure the creation of an effective application that leverages human language understanding. We will take a look at each step to comprehend it. [11]

1. **Data Acquisition:**

In this stage, we aim to acquire a dataset that reflects the linguistic diversity and complexity relevant to the application's subject. This step forms the foundation for understanding language and its varied usage.

1. **Text Cleaning:**

After obtaining the data, it's time to clean it from linguistic errors and distortions. This step is crucial to ensure accuracy and quality of results in later stages.

1. **Pre-processing:**

This part involves an initial analysis of the data and the application of basic natural language processing techniques. This step is fundamental to comprehend the language structure and prepare the data for training phases.

1. **Feature Engineering:**

This step focuses on utilizing features derived from the data in a way that contributes to the understanding of models and enhances their performance.

1. **Modeling:**

Modeling encompasses the training of models using processed data, refining them to achieve greater efficiency in understanding and processing language.

1. **Evaluation:**

After model training, their performance is evaluated using various tests to ensure their effectiveness and robustness in dealing with a wide range of scenarios.

1. **Deployment:**

Following model evaluation, the application is deployed for widespread use and application in real-world contexts.

1. **Monitoring and Model Updating:**

In this step, the application's performance is monitored post-deployment, and models can be updated based on changes in data or usage requirements.

### ML application Development

Developing machine learning applications is an exciting challenge in the modern tech world. These steps represent the journey developers go through to build smart applications based on the learning capabilities of automated systems. We’ll overview the main steps in developing these applications and how to achieve effective integration between the process components.

1. **Data Acquisition:**

Developers begin by examining and gathering data that the system will learn from. This data forms the foundation for understanding the application's context and training the model.

1. **Data Analysis and Cleaning:**

Developing machine learning applications requires a thorough examination of the data, ensuring its cleanliness and freedom from errors to ensure quality training.

1. **Feature Selection:**

Identifying key features in the data that will contribute to the system's understanding of information and achieve outstanding performance.

1. **Data Splitting:**

Dividing the data into sets for training and testing ensures an effective evaluation of the model and avoids overfitting problems.

1. **Model Selection:**

Modeling encompasses the training of models using processed data, refining them to achieve greater efficiency in understanding and processing language.

1. **Model Training:**

The model is trained using data, learning and benefiting from patterns and details within the data.

1. **Performance Evaluation:**

The model's performance is assessed using test data to ensure its effectiveness and efficiency in dealing with new cases.

1. **Improvement and Adjustment:**

Based on the evaluation, developers adjust enhance the model's performance and efficiency in facing different challenges.

1. **Application Deployment:**

After ensuring the model's quality, the application is deployed for public use and interaction with users.

1. **Performance Monitoring and Updating:**

Developers continue to monitor the application's performance after deployment, making necessary updates to keep up with changes and ensure continuous improvement.

## Similar Tools

### Software Requirements Classification Using Machine Learning Algorithms

Text classification is an attempt to organize documents based on specific properties and features. This supervised learning task involves determining categories for new documents based on training with a set of previously categorized documents. Software requirements are classified into two types: functional requirements and non-functional requirements.

**Methodology:**

Five steps are employed to perform software requirements classification, as illustrated in the following figure.

A diagram of a process

Description automatically generated

Figure 2: Phases of requirements classification pipeline.

[10]

The language used is Python, and the NLTK library was used, and the database is PROMISE\_exp, consisting of 969 requirements with their configurations represented as follows:



Figure 3: Number of requirements per label.

[10]

1. **Normalization**

In the first phase, we conducted the normalization process by converting all words to lowercase. Subsequently, we removed words with little or no significance, such as in the transformation from "The system shall refresh the display every 60 s” to “System shall refresh display every second”. Additionally, words were transformed into their root form, for example, "users" changed to "user".

1. **Feature extraction**

Both BoW and TF-IDF were used for feature extraction, and a comparison was made during the classification stage to observe which one led to an improvement in algorithm performance. The table below displays the top ten words that received scores for each technique. It can be noted that the words are the same, but the difference in importance starts from the third position.



Figure 4: Top 10 most important features.

[10]

1. **Feature selection**

After the feature extraction stage, we move on to feature selection, involving a filtering process to remove less important features using statistical methods such as CHI squared. Two parameters were utilized: the first one, max\_df, ignores terms with a frequency higher than the specified threshold, while the second one, max\_df, disregards terms with a frequency lower than the specified threshold.

1. **Classification**

Four algorithms were used for training and testing performance: LR, MNB, SVM, and K-NN. In LR, MNB, and SVM algorithms, a parameter called class\_weight was employed, which utilizes class label values to automatically adjust weights inversely proportional to the class frequencies in the input data. All hyperparameters for the classification algorithms were selected using a function called GridSearchCV, which tests all possible parameter combinations and returns the set that achieved the best results.

The scikit-learn tool was utilized, as it integrates various machine learning algorithms that were selected because it contains the algorithms used.

1. **Performance measure**

The data was divided into 10 folds, where 9 folds were used for training the algorithm, and 1 fold was used for testing. [10] The following performance measures were employed:

True positives (TP), false positives (FP).

True Negatives (TN), False Negatives (FN)

The F1 or F-measure combines precision and recall into a single metric. The F1 score is the harmonic mean, giving much more weight to low values compared to the regular average, which treats all values equally. The result is not high unless both recall and precision are elevated. [10]

**Results:**

The metrics used to evaluate the mentioned machine learning algorithms are Precision Equation, Recall Equation, and F-measure Equation, and the results of the evaluation are presented in the following figure:



Figure 5: Results of binary classification (FR and NFR).

[10]



Figure 6: Results of NFR classification.

[10]



Figure 7: Results of FR and NFR classification (12 granularities).

[10]

The results showed that the combination of TF-IDF and LR gives the best performance in binary classification, non-functional requirements classification, and general requirements classification. The F-value was 91% in binary classification, 74% in 11-category classification, and 78% in 12-category classification. [10]

### From user requirements to UML class diagram

This scientific paper focuses on extracting separation schemes from text requirements using NLP technologies and domain ontology. To analyse a given text, the most Natural Language Processing (NLP) systems are based on the following levels: Morphological level, lexical level, syntactic level, semantic level, discourse level and pragmatic level. Ontology is a branch of computer science and information science that is interested in studying how knowledge and information and their relationships are represented in a given field. [12]

This study accepts, as an input, textual data expressed in natural language and representing the user needs then identify the classes’ names, their attributes, and associations between them in order to classify them in a structured XML file. [12]

**The framework used:**

GATE is an open source framework developed using Java programming language used in this scientific paper. It is used to develop software components that process natural language. It can provide a set of natural language analysis tools that can take text inputs in English and as a result give basic forms of words and their parts of speech, etc., determine the structure of sentences in terms of phrases and dependencies of words, and state which name phrases refer to the same entities. [12]

GATE has an information extraction system (IE) called ANNIE (The New Quasi Information Extraction System) which contains many language processing as stated in this paper [12]:

* **Sentence splitter:**

the sentence splitter separates each sentence from the input string and returns a list of strings. [12]

* **Tokenizer:**

the tokenizer takes each sentence as an input and splits them into tokens such as words and punctuation. [12]

* **Syntactic parser:**

transforms sequences of words into structures indicating how sentence units relate to each other. This step helps us identify the main parts of a particular sentence such as the object, subject, verb...etc. [12]

**System architecture:**

**A diagram of a language analysis block

Description automatically generated**

Figure 8 DC-Builder System architecture.

[12]

Figure 8 shows DC-Builder System architecture. The natural language analysis block processes the requirements descriptions submitted by the user using the framework GATE, and specially: Sentence splitter. [12]

The classes of candidates can then be extracted by considering the name phrases in the text of the requirements. Candidate relationships can be found in the same way by considering verb phrases. For example, by analysing the phrase "the doctor gives medicines to the patient" we can know three candidate classes (doctor, medication, patient) and one candidate relationship (give). In this context, heuristics can play a fundamental role to facilitate such task [12].

Given a parts-of-speech and their functions in sentences, Chen [13] proposed eleven rules in order to translate NL requirements description written in English from natural language (English) to an entity-relationship diagram. The proposal of Chen seems to be the first attempt using linguistic. [12]

**The rules described below will be used to facilitate the extraction of classes’ names for the diagram:**

* All nouns are converted to entity types. [12]

It can be concluded that all nouns can be mapped onto the names of the classes’; By nouns, we mean all types of nouns such as common nouns and collective nouns. [12]

* A gerund may indicate an entity type which is converted from a relationship type.

Firstly, A gerund can be defined as a noun which consists of a verb and an “ing”. It is often called an -ing word or a verbal noun. [12]

* a specialization’s relationship between entities: sentence’s structure “is a” can relate two nouns A and B to one another. [12]
* every proper noun (Person name, Location name …) is ignored to be a class. [12]

This rule can help to perform a partial filtering in order to obtain an accurate set of classes’ names for the class diagram. [12]

For attributes extraction, there some heuristics:

* A noun such as “vehicle\_number”, “group\_no”, “person\_id” and “room\_type” may refer to an attribute type. [12]
* A noun phrase succeeding the “has/have” verb phrase may indicate the presence of attribute types. [12]

For associations’ extraction, there Three heuristics:

* A transitive verb can be a candidate for relationship type. [12]

Transitive verb, in syntax, is a one that requires an object to complete its meaning. [12]

* if a verb is equal to one of the following list {“include”, “involve”, “consists of”, contain, “comprise”, “divided to”, “embrace”}, therefore, this relationship can be aggregation or composition. [12]

After that as an input, the concerned module produces an initial XML file that should be refined. [12]

**Refinement using domain ontology:**

The previous module produces an initial model, in XML form, including concepts related to classes, attributes, and associations. This model can contain erroneous elements which should be treated. In fact, the constructed ontology will help eliminate irrelevant elements. [12]

### AI Based UML Diagrams Generator

This study will focus on providing solution to generate Use-Case diagram and Class diagram against the particular business requirement. system will read and understand the business requirement using Natural Language Processing and Machine learning and identifying entities and relationships on that for generate use case diagram. And also, classes and relationship between classes will also be identified to generate class diagram. Then system will show generated diagram with including user interacting feature where user can add additional element and add changes to use case or class diagram or edit existing element and its attribute or relations. [13]



Figure 9 Proposed Solution Design

As showing in the Figure 6 proposed solution design, user can upload a text file containing a scenario or simply copy and paste the text inside. The system extracts word fragments and named entities using the NLP module. Subsequently, the ML module will identify and extract the usecases, actors, classes, associations according to the ML algorithms which have implemented. Plant UML modeling draws the use case and class diagram. The system allows editing of the generated diagrams and the user can modify and adjust the use cases, actors, classes, attributes and all types of relationships. Therefore, the output is a highly customizable output that the user will be able to achieve the desired output according to the business requirements provided. [13]

**Methodology**

The text assure the importance of efficiently extracting information from text data, noting that manual text searching can be difficult and time-consuming, which may lead to important information being skipping. Therefore, providing an automated way to extract information from texts and present it in an organized manner can provide many benefits and save time spent browsing text documents. Information mining technology aims to achieve this goal by systematically analyzing texts and identifying relevant information. The example provided assure the importance of understanding the parts of speech in a sentence to accurately interpret the meaning. [13]

**Algorithmic design**



Figure 10 Process flow with NLP And ML Model

Figure 6 shows the process flow with Natural language processing and Machine learning model where several preprocessing stages has involved before proceeding to the ML based information extraction. [13]

The important information in requirement texts usually follows certain patterns. When we look at the Parts of Speech tagging of a sentence, they follow specific patterns too. If we understand these patterns, we can easily get the information we need. This helps us create use case and class diagrams for the project. [13]

In algorithmic design, we can choose any Parts of Speech tagger modules for English literature. We mark words in a story based on their meanings and how they're used in specific parts of speech. [13]

Some POS tagging examples can be mention as: Coordinating Conjunction (CC), Cardinal Digit (CD), existential (EX), adjective (JJ), modal (MD) such as could, will etc., proper noun (NNP), predeterminer (PDT), possessive pronoun (PRP), etc. (Rachiele, G.2018) To assign grammatical information to each word in a sentence we used POS taggers. [13]

Most of the information in a requirement text is made up with verb, gerund/present (VBG), noun plural (NNS), TO, verb base (VB), noun (NN), preposition or subordinating conjunction with these pos taggers. But they come with different patterns. [13]



Dataset would be like as Figure 7 shown below according to the above requirement text.



Figure 11 Sample dataset

For now, according to the given sample, there are 3 kinds of specific patterns that can identify:

• {VBG>NNS>TO>VB>NN}

• {NN>IN>DT>NN}

• {VB>DT>NNS>TO>VB}

The process involves finding the key terms for class or use case diagrams using machine learning, which makes this task a classification one. Data is divided into two sets: training and testing. The trained model uses features from the training instances to predict terms for unseen data. Comparing predicted and actual terms helps evaluate the model's performance. Statistical techniques rely on adjustments between classification results using probability and costs for decision-making. [13]

Preprocess Functional requirement texts

The text discusses the process of text preprocessing, where ambiguity resulting from the use of multiple forms of a certain verb or the singular/plural form of a word is reduced. Additionally, common words like "a," "the," "of," and "is" do not contribute much information to our summarization goal. Below are several operations commonly used for document preprocessing. [13]

**Document segmentation:**

A text is divided into several paragraphs to find where each sentence is placed in its respective paragraph. [13]

**Stemming**:

We apply stemming to bring a word to its root or base form. The examples include use of a singular form rather than using a plural, or removal of ‘ing’ from a verb. To this end, Stanford NLP stemmer is employed in this paper. Paragraph segmentation. Paragraph segmentation divides a paragraph into sentences using sentence tags. [13]

**Word normalization:**

Each sentence consists of multiple normalized words. Through normalization and lemmatization individual words become one common form, stemming down to their roots. Ambiguities are removed by Porter’s algorithm. [13]

**Stop word filtering:**

Stop words can be filtered out after performing other preprocessing steps. There is no uniform rule for selecting a stop word because it depends on individual tasks. In this work, words such as a, is, in, the, of are selected as stop words and are filtered out from the document. In text mining applications stop word filtering is considered as a standard step. [13]

Machine Learning Techniques to Identify Diagram Elements

1. Classification Model
2. Recurrent neural network Unit
3. RNN based Sequence to Sequence Model
4. Strategies of training

**IMPLEMENTATION:**

The initial dataset is divided into subsets containing functional requirement sentences for each element in the diagram, specifically actors, use cases, and classes. Each subset is further divided into validation, training, and test pairs. Separate machine learning models are built for each diagram element using the training subsets. During implementation, data preprocessing is conducted using Python programming, which involves cleaning and preparing the data for analysis. [13]

check the words in the English dictionary and removing unused punctuation, removing stop words etc. are the preprocessing stages done in the implementation.

Thereafter extracting feature task was perform where it checks existence of whole keyword and also existence of all parts of known keywords were performed.

The recurrent words such as propositions don’t participate as key terms. Calculating the term frequency and inverse document term frequency (TF X IDF), these expressions can dominate other meaningful expressions. Consequently, to deal with some challenging terms we can eradicate from script by allocating probability assessment of zero and also, numerals and nonalphanumeric characters will be removed.

Thereafter naïve based classifier has been implemented and as previous stages clean and tokenization tasks also perform before generate predicted tags based on decision rule

In the decision rule method, it takes the functional text, and the keywords from the early stage perform and also stopwords. First it gets scores of each whole keywords from the functional text provided. Then get scores for each known keywords with all parts in the requirement sentence. After that, three steps performed as.

1. adding tag if posterior probability is higher than or equal to 0.5
2. add highest scoring to TF-IDF keywords if not already added.
3. Add ‘c#’ if there no tags (‘c#’ just a special keyword to identify it clearly from other keywords)

**Identifying Use case and Class Diagram Relationships with NLP and ML techniques**

In this implementation our focus mainly on identifying relationship in both use case and class diagram related components separately. This is a combination of two models where one model will be identifying relationship involved entities. As an example of use case diagram first model will identify what are the actors and use cases involve in the relationship. Second model will be identifying the relationship type of that particular relationship. These two models need to build for both use case and class diagram separately. Two models per each diagram type to identify relationship involved in particular functional requirement text. [13]

Similar to the previous implementation, in this iteration, we again divide the initial dataset into subsets containing functional requirement sentences for each diagram element. This time, we focus on use case relationships and class relationships. Each dataset is then separated into two subsets for the two models: one with entities involved in the relationship and the other with the relationship type along with the functional text. These subsets are divided into validation, training, and test pairs, with the same distribution as before. Two separate machine learning models are built for each diagram type using these training datasets. [13]

**Draw diagram with identified diagram elements**

With this implementation we want to generate the usecase and class diagram according to identified elements through previous two implementation. Usecase and class diagram elements that has been identified from functional requirement text has outputted as JSON file format. [13]

**Evaluation approach**

To evaluate the Model, we can use the term called Model Evaluation Metrics.

Model evaluation metrics are needed to measure model performance. The choice of evaluation parameters depends on the machine learning task provided. In this project that is classification. [13]

**Classification metrics**

When performing predictions through classification, four kinds of result that could occur as follow. [13]

* True positives: are when we assume an observation belongs to a particular class and actually it is belonging to that exact class.
* True negatives: are when we assume an observation is not belonging to a particular class and that actually not belonging to that particular class as expected.
* False positives: occur when we predict our observation belongs to a one particular class but in reality, it is not belonging to that class.
* False negatives: occur when we assume an observation is not belonging to a particular class but actually it is belonging to that same class



*Figure 12 confusion matrix for the case of binary classification*

Accuracy, precision, and recall are the three main metrics that can be used to evaluate a classification model in generally. [13]

1. Accuracy is the percentage of correctly predicted instances in the test data, calculated by dividing the number of correct predictions by the total number of predictions.
2. Precision is the fraction of true positives (relevant examples) among all examples predicted to belong to a particular class, indicating how many relevant items were predicted.
3. Recall is the fraction of test data predicted to belong to a particular class compared to all test data truly belonging to the class, indicating how many relevant items were predicted.

## Summary

In this chapter thoroughly examines the research context, starting with a detailed review of existing literature. The emphasis is on a meticulous analysis of Very precise. The research methodology, which includes Google Scholar, digital libraries for data collection, is outlined succinctly.

The chapter then delves into the domains of Artificial Intelligence (AI), covering its definition, conduct, and techniques such as knowledge representation, search, automated reasoning, and planning. Machine Learning (ML) is introduced as a branch of AI, focusing on supervised learning, neural networks, and their applications like NLP.

The discussion on NLP includes text tokenization, word embedding, and techniques like TF, TF-IDF, and Word2Vec. Named Entity Recognition (NER), parsing, pronoun resolution, and semantic analysis are explored in the context of NLP.

The chapter transitions to system development for AI, outlining the stages for developing NLP and ML applications. Python and various libraries are highlighted for development. Similar tools in the field, such as Lucidchart, Visual Paradigm, StarUML, Enterprise Architect, and MagicDraw, are introduced, emphasizing their AI-powered features.

After investigating these topics, we will be prepared in Chapter 3 to analyze the system and find both functional and non-functional requirements.

# Chapter 3: System Analysis

## Introduction

In this chapter, we will focus on defining the system requirements using simple diagrams. We will use UML tools to explain how the system operates and its requirements. We will specify both functional and non-functional requirements and discuss the development methodology. We will also analyze an existing system.

## Methodology

### SDLC

In our project, we used the waterfall methodology for project execution. In the first step, the project plan. In the second step Literature Review. in the third step, analyzed the system. in the fourth step, system design.

Stability of Requirements: The Waterfall model requires defining project requirements early. It fits well when requirements are stable and undergo minimal changes.

Path Determination: The Waterfall model follows a linear and organized approach, allowing the team to clearly outline the workflow and task sequence. This structure is beneficial for projects requiring precision and strict timelines.



Figure 13: waterfall for project.

### Analysis

In the analysis phase we have identified 4 basic activities to fully analyze the components of the system. Before starting with system design in general we must collect the requirements and analyze them correctly. During the analysis phase of the system, we used this existing strategy in Figure 16.



Figure 14: Description of stages of analysis

In the process of systems analysis existing, we will compare work related to our system and identify points of difference between them and mention the advantages of each work like our system.

In the process of requirements elicitation, we will describe system requirements and functional and non-functional requirements in the natural language, and we will Elicit them through brainstorming, use similar systems requirements, use the Lucid platform tool to generate ideas for the requirements.

In the process of requirements specification, we will construct tables to describe both functional and non-functional requirements.

In the process of requirements modeling, we will draw use cases that show all possible interactions with the system and write the description for them.

## Analysis of Existing Systems

At this stage, we compared our system with similar systems through features. Then we used some of the features of similar systems at the stage of the elicitation requirements.

The checkmark  represents that the advantage will be achieved in the system whereas the  shows that it will not be achieved in the system. “**?**” it shows that it is possible to try to make a feature in the system. The differences between our system and other comparable work are shown in Table 3.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| System  Feature | AI Based UML Diagrams Generatorrm | From user requirements to UML class diagram | Software Requirements Classification Using Machine Learning Algorithms | Our project |
| Create UML diagrams |  |  |  |  |
| Code Generation |  |  |  |  |
| Automated generation of drawings. |  |  |  |  |
| POS tagging |  |  |  |  |
| Edit model |  |  |  |  |

Table 3: Related Work Comparison

## Requirements Elicitation

Requirements engineering is the process of discovering all system services, restrictions imposed on it, and services that meet user needs, analyzing, documenting, and verifying these services [14]. It includes high-level activities, including elicitation, analysis, and other activities. Requirements elicitation is the collection of all requirements, including system requirements and user requirements [14]. System requirements describe all the requirements that we want the system to perform [14]. After the elicitation process, we extracted the system requirements, and from them we identified the functional and non-functional requirements through brainstorming, using similar requirements tools, and using the Lucid platform tool.

### System Requirements

A structured document setting out detailed descriptions of the system’s services and operational constraints [15].

Defines what should be implemented so may be part of a contract between client and contractor [15].

* 1. To build an AI website system to receive software development requirements texts and documents and use NLP techniques to analyse them.
  2. To extract classes, functions (methods / procedure) and relations between classes and function.
  3. To generate UML diagrams: use-cases and class diagram.

### Functional Requirements

The requirements determine the system's behavior, what it should and shouldn't provide, and how it interacts with inputs [14].

1. To build an AI website system to receive software development requirements texts and documents and use NLP techniques to analyses them.
   1. The system shall allow the user to enter requirements either by typing directly or by uploading a document.
   2. The system shall allow the user to create analysis model.
   3. The system shall allow the user to choose type of UML diagram such as (class, use case).
   4. The system shall prepare to clean and organize the initial data to perform the text analysis process.
   5. The system should analyze the text by doing Tokenization, Parsing and Relation Extraction.
2. To extract classes, functions (methods / procedure) and relations between classes and function.
   1. The system shall analyses the requirements context using NLP techniques.
3. To generate UML diagrams: use-cases and class diagram.
   1. The system shall draw the diagram after generated by NLP techniques.

### Non-Functional Requirements

It describes the characteristics of the system and the constraints imposed on the services provided by the system such as time constraints and other constraints. It also describes specific standards for some of the system's services [14].

1. To build an AI website system to receive software development requirements texts and documents and use NLP techniques to analyses them.
2. The system must be fast, ensuring that the processing of requirements and drawing (use case or class diagram) does not exceed (write the time).
3. The resulting diagrams in the system should have an accuracy of no less than (write the percentage).
4. It should be able to handle (write the range of users) simultaneously.
5. The system should be user-friendly and adaptable to various devices to provide a seamless experience.
6. The system should perform its functions with minimal steps.
7. To extract classes, functions (methods / procedure) and relations between classes and function.
8. The system should be able to recognize and analyze a minimum of (write the percentage) of user requirements entered, with a permissible margin for error not exceeding (write the percentage).

## System Scenario

A black and white rectangular object with text

Description automatically generated

## Requirements Specification

A Use case represents a class of functionality provided by the system. Use case can be described textually, with a focus on the event flow between actor and system [17]. The textual use case description consist of 6 parts:

* Unique name
* participating actors
* Entry condition
* Exit condition
* Flow of event

### Usecase 1:

|  |  |
| --- | --- |
| analysis documents | Use case |
| User | Actor |
| 1. user entered the requirements. 2. analysis documents: 3. Elementsfinder 4. Relationshipfinder : 5. After analysis is completed generate Diagram Specs get the analysis like number the class and the relation between them and then ready to draw. | Description |
| Data is available for analysis | Pre-condition |
| the analysis is completed | Postcondition |

Table 3: use-case (analysis documents)

### Usecase 2:

|  |  |
| --- | --- |
| Enter requirements Document | Use case |
| User | Actor |
| 1. The user clicks on the text box. 2. The user enters requirements by typing.   Or   1. The user clicks on the select doc. 2. Choose the doc | Description |
| The user enters the website. | Pre-condition |
| The requirements are entered successfully | Postcondition |

Table 4: use-case (Enter requirements Document)

### Usecase 3:

|  |  |
| --- | --- |
| create analysis model | Use case |
| User | Actor |
| 1. The user clicks on select type. 2. The system shows to type of model (class model, use-case model)   If the user chooses class model   1. The system shows class model on screen after the user start generate.   If the user chooses class model   1. The system shows use-case model on screen after the user start generate | Description |
| The system will determine whether it is the context or Document to Draw the Diagram. | Pre-condition |
| The selected model type is displayed on the screen. | Postcondition |

Table 5: use-case (Create analysis model)

### Usecase 5:

|  |  |
| --- | --- |
| use-case diagram | Use case |
| User | Actor |
| 1. After analysis completed the system draws a use-case model after the requirements. 2. The system displays a use-case model on screen. | Description |
| The system is ready and prepared to start drawing the diagram | Pre-condition |
| The diagram has been successfully drawn by the system. | Post-condition |

Table 6: use-case (Draw use-case model)

### Usecase 6:

|  |  |
| --- | --- |
| Use case | class diagram |
| Actor | User |
| Description | 1. The system draws a class model after the analysis requirements are completed. 2. The system displays class models on screen. |
| Pre-condition | The system is ready and prepared to start drawing the diagram |
| Post-condition | The diagram has been successfully drawn by the system. |

Table 7: use-case (Draw class model)

## Requirements Modeling

### usecase



Figure 15: user use-case diagram

The user can enter requirements documents as text or a file and then choose Model type (class , usecase)Analysis documents by Find Elements and Find Relation After analysis is completed generate Diagram Specs get the analysis like number the class and the relation between them and then draw.

### Class Diagram

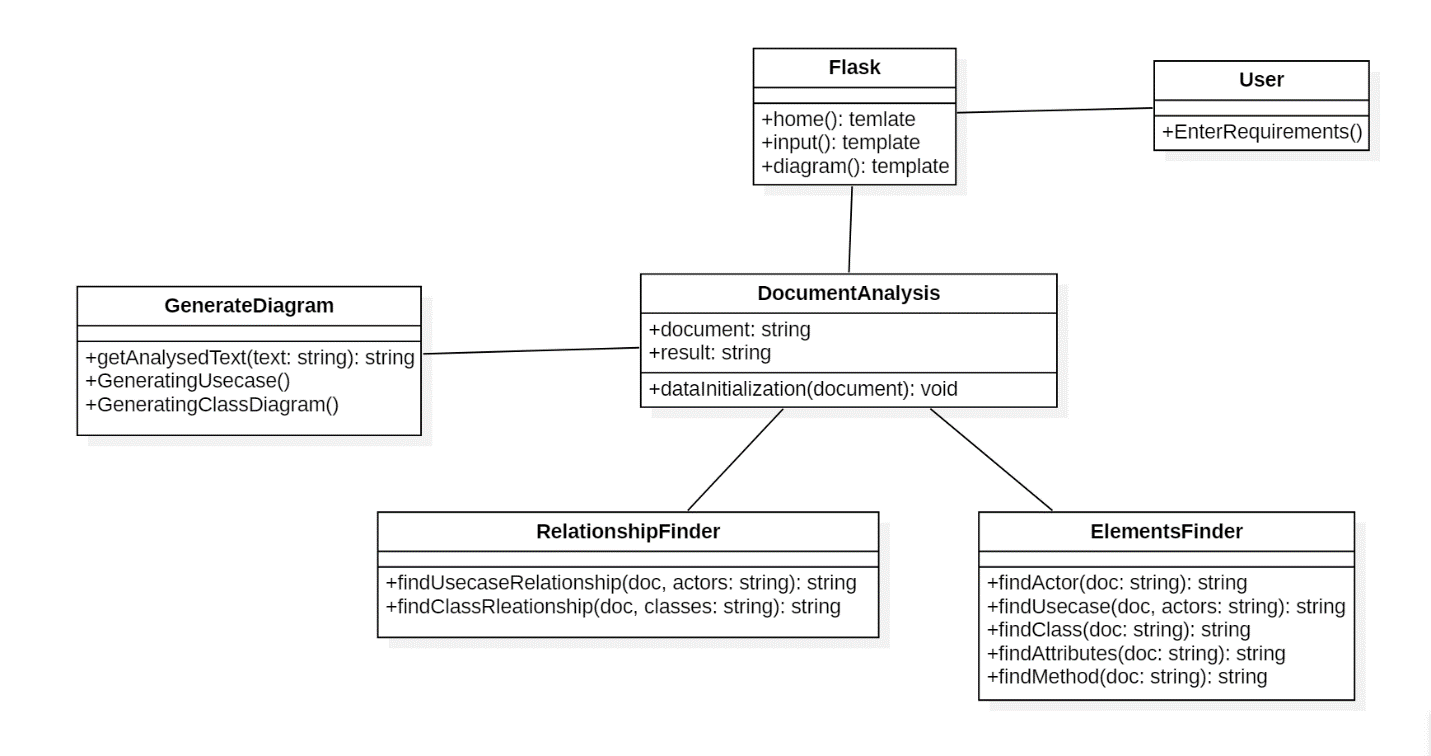


Figure 16: class diagram for our system.

The User class is where the user’s functions reside, such as inputting requirements.

The Flask class acts as an intermediary between the user and our system, providing the graphical interface (HTML CSS) and sending information to the DocumentAnalysis class.

The DocumentAnalysis class prepares the input data and then sends it to the ElementsFinder and RelationshipFinder classes.

The ElementsFinder class finds elements of the diagrams such as actors, use cases, classes, and others.

The RelationshipFinder class finds relationships in the use case and class diagrams.

The GenerateDiagram class draws the diagrams after analyzing the data.

## Summary

In this chapter, we analyzed the system, identified the methodology of analysis and development, identified the functional and non-functional requirements, and displayed them in the format of tables and UML diagrams such as use case and class diagram.

After analyzing the system, we will be ready in chapter 4 to design the system and determine the architecture of the system and design the user interface.

# Chapter 4: System Design

## Introduction

This chapter covers system design, including structural design, component design, data modelling, and user interface design. Structural design focuses on organizing the system's structure to ensure efficient performance, while component design addresses the interaction of software components to achieve functional goals. Data modelling design deals with efficiently organizing and storing data, and user interface design highlights improving the user experience through an effective and attractive interface.

## Design Methodology



Figure 17:Design methodolgy

**Architectural Design:**

The process of determining the overall structure and organization of a software system, involving high-level decisions on system configuration, key component specification, and their interactions.

**Component Design:**

The process of dividing a system into manageable units, with each unit representing a specific module or function within the software.

**Data Modeling Design:**

The process of defining and organizing data requirements for a system, creating a conceptual representation of data, defining relationships between entities, and specifying data storage and access.

**User Interface Design:**

The process of creating an easily understandable and visually appealing interface for users to interact with software, including the design of structure, navigation, and visual elements. Goals include enhancing user experience through user-friendly and efficient interface design.

## Architectural Design

The architectural design is concerned with understanding how a system is organized and designing the overall structure of the system, it involves identifying major system components and their communications.



Figure 18: Architectural Design

We opted for the client-server architecture because it allows for the distribution of tasks between the client and the server. In our system, on the customer side, the user enters a file with requirements or text requirements that are entered via the keyboard. After that, user requests are received through the user interface and sent to the server. After processing the requirements in the server and sending the generated diagram to the user interface at the client, the user interface displays the diagram to the user. Then the user can download the graphic in PDF format.

On the server side, the server sends the file or user request to the NLP class, which processes the requirements and analyzes the texts using the SpaCy library. A UML file is created based on the results extracted from the text analysis, and then the file is sent to the PlantUML tool to convert it into an image. Finally, the server sends the file generated by UML to the client.

## Component Design

Our system is divided into two main parts: document analysis and diagram drawing. Each part contains several categories. The document analysis part includes finding elements of use cases and class diagrams and identifying relationships. The diagram drawing part includes drawing an analysis model, a use case diagram, and a class diagram.

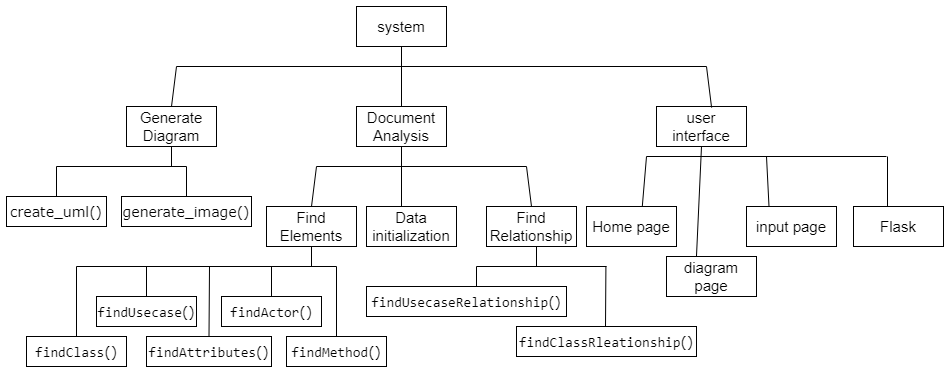


Figure 19:Component Diagram

### First main component: document analysis

**class ElementsFinder:**

**method:**

1. **findActor(doc):**

This method aims to find actors (entities) in the text. It searches for nouns in the document (doc) and identifies them as actors if they meet certain conditions. It considers a noun as an actor if it's preceded by "the" or followed by a verb, auxiliary verb, or adverb. Then, it looks for verbs that have nouns directly as children, and it returns the common actors found by both methods.

1. **findUsecase(doc, actors):**

This method identifies use cases based on the verbs associated with the provided actors. It iterates through the document and looks for verbs that have nouns (actors) as children. If the child noun matches any of the provided actors, it considers it a use case. Additionally, it checks for direct objects (dobj) of these verbs and appends them to the use case description. It handles conjunctions like "and" or "or" to create more descriptive use case names.

1. **findClass(doc):**

This method simply uses findActor() to find actors in the text and considers these actors as classes in the text.

1. **findAttributes(doc):**

This method identifies attributes associated with actors in the text. It looks for nouns or adjectives that come after pronouns or particles in the document and considers them as attributes.

1. **findMethod(doc, actor):**

This method identifies methods associated with the provided actors in the text. Similar to findUsecase, it looks for verbs with nouns as children, checks if the noun matches any of the provided actors, and extracts any direct objects of these verbs to construct method names. It also handles conjunctions to create descriptive method names.

**class RelationshipFinder:**

**method:**

1. **findUsecaseRelationship(doc, actors):**

This method takes a document (doc) and a list of actors as input parameters. It iterates through each token in the document and checks if the token's part of speech is a verb. If it is, it iterates through the token's children to find nouns. If any of these nouns match with the actors provided, it creates a relationship string in the format "noun --> verb" and appends it to the relationship list. Finally, it returns the list of relationships found.

1. **The findClassRelationship():**

### First main component: Generate Diagram:

This class generates an image from the UML diagram text using PlantUML. It, which runs PlantUML with the diagram.uml file as input and specifies the output.

**method:**

1. **create\_uml()**
2. **generate\_image()**

**user interface:**

It sends the requirements entered by the user, receives the diagram, displays it to the user and allows the user to download the diagram in PDF format. The user interface has 3 pages which is home page, entry page and diagram display page. The user interface was constructed using the Flask framework.

## Data Modeling Design

The program converts the text to lowercase, then divides it into sentences and stores each sentence in a row called 'requirements'. After that, the program outputs various elements such as 'actor', 'use case', 'relation between actor and use case', 'class', 'method', 'attribute', and 'relation between classes'. Finally, it creates a file with the extension 'diagram.uml' to draw the digram.

And we used this dataset.

• Software requirement dataset can be find in keggle

(<https://www.kaggle.com/iamsouvik/software-requirements-dataset>) .

• Labeled requirement dataset can be find in zenodo

(<https://zenodo.org/record/268542#.X05hwMgzZhF>) .

• Freely available Natural Language Requirements Dataset

(<http://fmt.isti.cnr.it/nlreqdataset/>) .

## User Interface Design



Figure 20: Home Page

1. The system should be user-friendly and adaptable to various devices to provide a seamless experience.



Figure 21: Input Page

1. The resulting diagrams in the system should have an accuracy of no less than (write the percentage).
2. It should be able to handle (write the range of users) simultaneously.
3. The system should be user-friendly and adaptable to various devices to provide a seamless experience.
4. The system should perform its functions with minimal steps.
5. The system should be able to recognize and analyze a minimum of (write the percentage) of user requirements entered, with a permissible margin for error not exceeding (write the percentage).



Figure 22: Diagram Page

1. The system must be fast, ensuring that the processing of requirements and drawing (use case or class diagram) does not exceed (write the time).
2. The resulting diagrams in the system should have an accuracy of no less than (write the percentage).
3. The system should perform its functions with minimal steps.
4. To extract classes, functions (methods / procedure) and relations between classes and function.

## Summary

In this chapter, we have created the Architectural design and utilized the client-server pattern. The component and data modeling design will be developed in the next phase. Finally, we have initiated the design of the user interface.

In the upcoming chapter, we will summarize everything we have accomplished since the beginning of the project. We will discuss how we achieved our goals, the lessons Learnt, and future work.

# Chapter 5: Implementation and Testing

In this chapter, we delve into the implementation and testing phase of the program, which is one of the most crucial stages in the software development process. We start by implementing the program using the Python programming language, where the code is written based on the design prepared in the previous chapters. Then, we move to the testing phase, which aims to verify the correctness and efficiency of the program through a variety of tests. We hope this chapter will be useful for understanding how to implement and test the program in an effective and systematic way.

## Methodology

A black and white label with black text

Description automatically generated

Figure 23: Implementation and Testing Methodology

**Implementation:**

In the implementation phase, the program code is displayed and explained in detail. This includes clarifying functions, variables, brackets, loops, and other programming elements. The aim of this display and explanation is to ensure a full understanding of the software, and to provide the ability to maintain and update it in the future. This also helps achieve transparency and clarity in the software development process, allowing anyone reading the code to easily understand it.

**Testing:**

In the testing phase, we will conduct two main tests to ensure the quality of the program. The first is a combined system and scenario test, where we check that all parts of the program work together correctly and that the program handles a variety of potential scenarios correctly. Lastly, we will conduct a requirements test, where we check that all program requirements have been correctly implemented.

## Implementation

In the implementation phase, the program design is transformed into actual code using the Python programming language. The code is displayed and explained in detail to ensure a full understanding of the software and to facilitate its maintenance and updating in the future. This phase helps achieve transparency and clarity in the software development process and ensures that the program code complies with standards and good practices in programming, which helps improve the maintainability and overall performance of the program.

A computer screen shot of a program

Description automatically generated

Figure 24: Showcases the findActor(doc) method in our system.

This function is designed to find the actors in a given document (doc). Here, an actor is defined as a noun that is either preceded by the word the or followed by a verb, auxiliary verb, or adverb. The function also considers nouns that are children of a verb as actors.

Here's a step-by-step breakdown:

* 1. The function takes a document (doc) as input. This document is expected to be a processed text where each word is tokenized, and its part-of-speech (POS) is identified.
  2. Two empty lists, actor1 and actor2, are initialized to store the "actors" found in the document.
  3. The function then iterates over each token in the document. For each token, it checks the following:
     1. If the token is a noun and the previous word is the, it adds the token to the actor1 list.
     2. If the token is a noun and the next word is a verb, auxiliary verb, or adverb, it also adds the token to the actor1 list.
     3. If the token is a verb, it checks all its children (words that are related to this verb in the sentence structure). If any child is a noun, it adds the child to the actor2 list.
  4. After going through all the tokens, it finds the intersection of actor1 and actor2 (i.e., the actors that appear in both lists) and stores it in finalActor.
  5. Finally, it returns finalActor, which is a set of "actors" found in the document according to the criteria defined above.

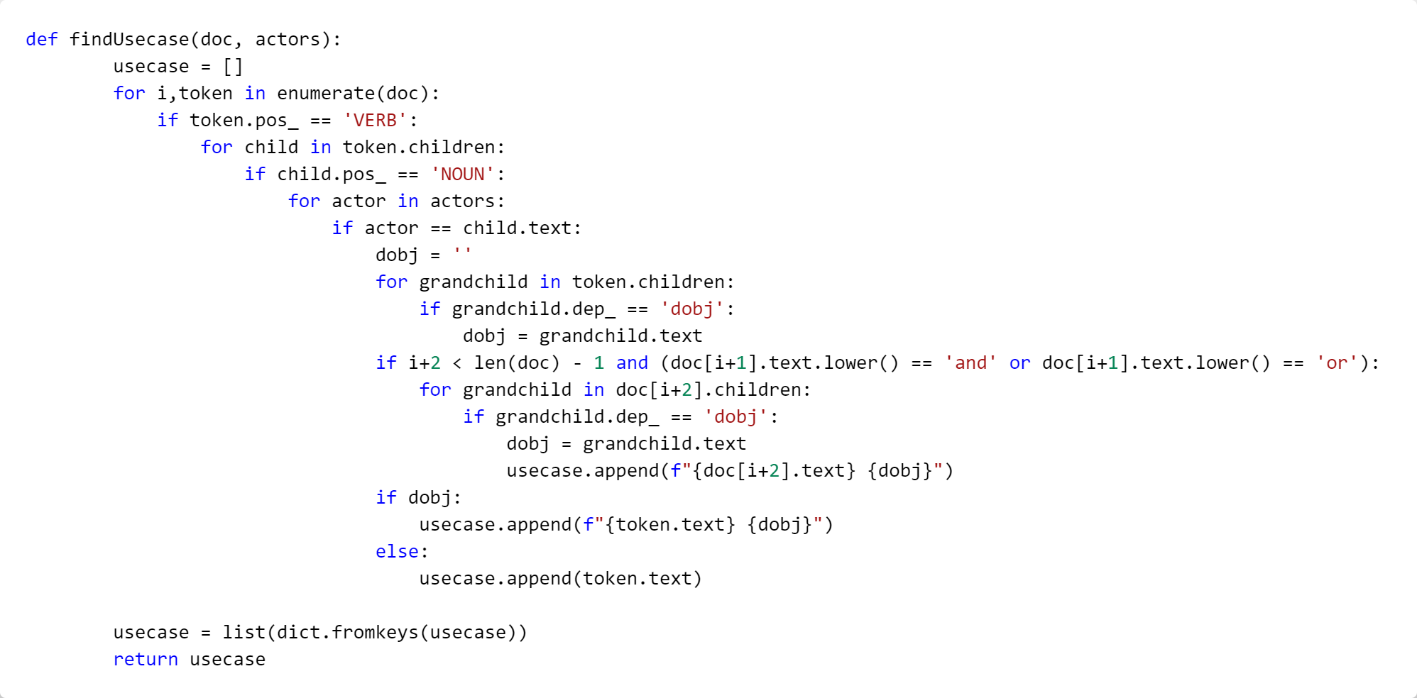


Figure 25: Showcases the findUsecase(doc, actors) method in our system.

This function, findUsecase, is designed to find the use cases in a given document (doc) based on the actors involved.

Here’s a step-by-step breakdown:

1. The function takes two arguments: a document (doc) and a list of actors (actors). The document is expected to be a processed text where each word is tokenized, and its part-of-speech (POS) is identified.
2. An empty list, usecase, is initialized to store the use cases found in the document.
3. The function then iterates over each token in the document. For each token, it checks the following:

The function checks if the token is a verb.

If the token is a verb, it checks all its children. These are words that are related to this verb in the sentence structure.

If any child of the verb is a noun and is also in the list of actors, the function begins to construct a use case.

The function initializes an empty string dobj to store the direct object of the verb (if any).

The function checks the children of the verb again. If any child is a direct object (dobj), it stores the text of the child in dobj.

If the current token and the next two tokens form a conjunction (i.e., the next token is ‘and’ or ‘or’), the function checks the children of the third token.

If any child of the third token is a direct object, it stores the text of the child in dobj and appends the verb and dobj to the usecase list.

If dobj is not empty, the function appends the verb and dobj to the usecase list. Otherwise, it just appends the verb to the usecase list.

1. After going through all the tokens, it removes duplicates from the usecase list by converting it to a dictionary and then back to a list.
2. Finally, it returns to the usecase, which is a list of use cases found in the document according to the criteria defined above. Each use case is a string that consists of a verb and possibly a direct object. The verb represents an action performed by an actor, and the direct object (if any) is the object on which the action is performed.

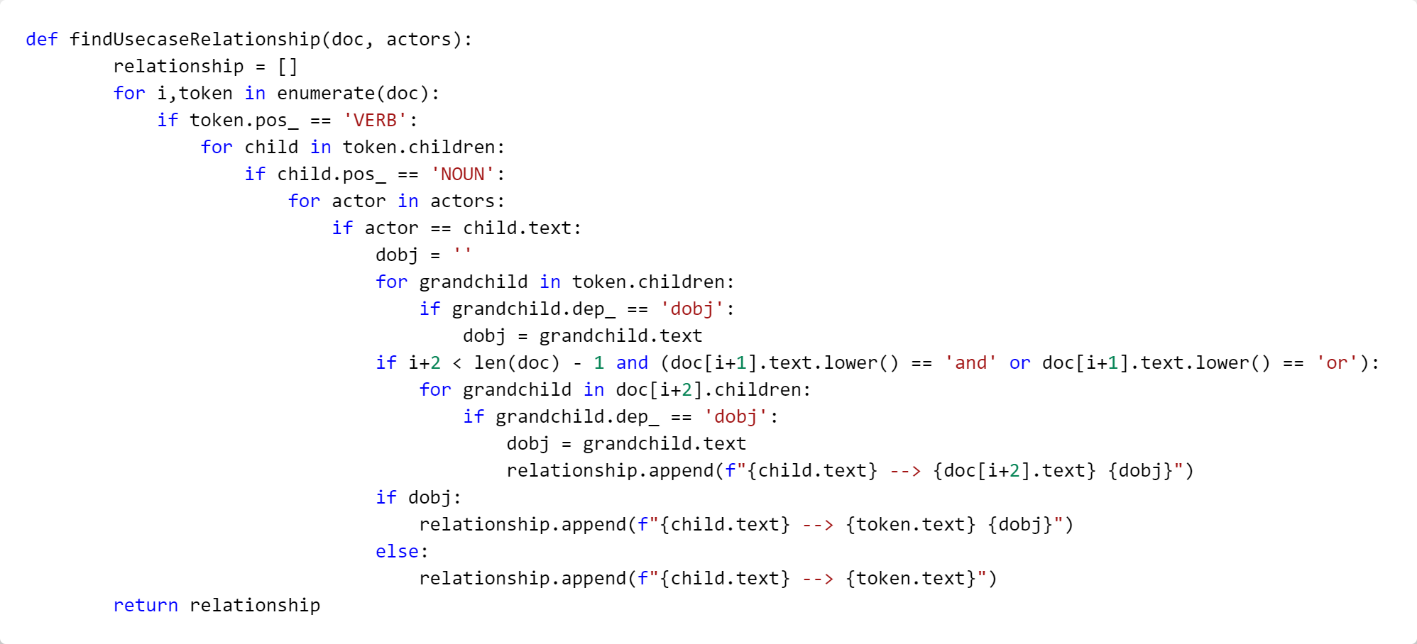


Figure 26: Showcases the findUsecaseRelationship(doc, actors) method in our system.

This function, findUsecaseRelationship, is designed to find the relationships in a given document (doc) based on the actors involved.

Here’s a step-by-step breakdown:

1. The function takes two arguments: a document (doc) and a list of actors (actors). The document is expected to be a processed text where each word is tokenized, and its part-of-speech (POS) is identified.
2. An empty list, relationship, is initialized to store the relationships found in the document.
3. The function then iterates over each token in the document. For each token, it checks the following:

If the token is a verb, it checks all its children. These are words that are related to this verb in the sentence structure.

If any child of the verb is a noun and is also in the list of actors, the function begins to construct a relationship.

The function initializes an empty string dobj to store the direct object of the verb (if any).

The function checks the children of the verb again. If any child is a direct object (dobj), it stores the text of the child in dobj.

If the current token and the next two tokens form a conjunction (i.e., the next token is ‘and’ or ‘or’), the function checks the children of the third token. If any child of the third token is a direct object, it stores the text of the child in dobj and appends the child’s text, the third token, and dobj to the relationship list in the format of a relationship (i.e., child.text --> doc[i+2].text dobj).

If dobj is not empty, the function appends the child’s text, the verb, and dobj to the relationship list in the format of a relationship (i.e., child.text --> token.text dobj). Otherwise, it just appends the child’s text and the verb to the relationship list in the format of a relationship with no direct object (i.e., child.text --> token.text).

1. After going through all the tokens, it removes duplicates from the relationship list by converting it to a dictionary and then back to a list.
2. Finally, it returns relationships, which is a list of relationships found in the document according to the criteria defined above. Each relationship is a string that represents a relationship between an actor and a verb (representing the action performed by the actor) and possibly a direct object (representing the object on which the action is performed).

A black text on a white background

Description automatically generated

Figure 27: Showcases the findClass(doc) method in our system.

The main reason for writing the code in this way, rather than just relying on the previous function, is for maintenance and continuous improvement. In case of any issues or if there is a need to modify the code, we can easily modify the outputs of the previous function without affecting the rest of the code.

In addition, we can also add some conditions or additional checks to increase the efficiency and overall performance of the code in the future. This helps improve the maintainability and flexibility of the code, making it more adaptable to changes or new requirements in the future.

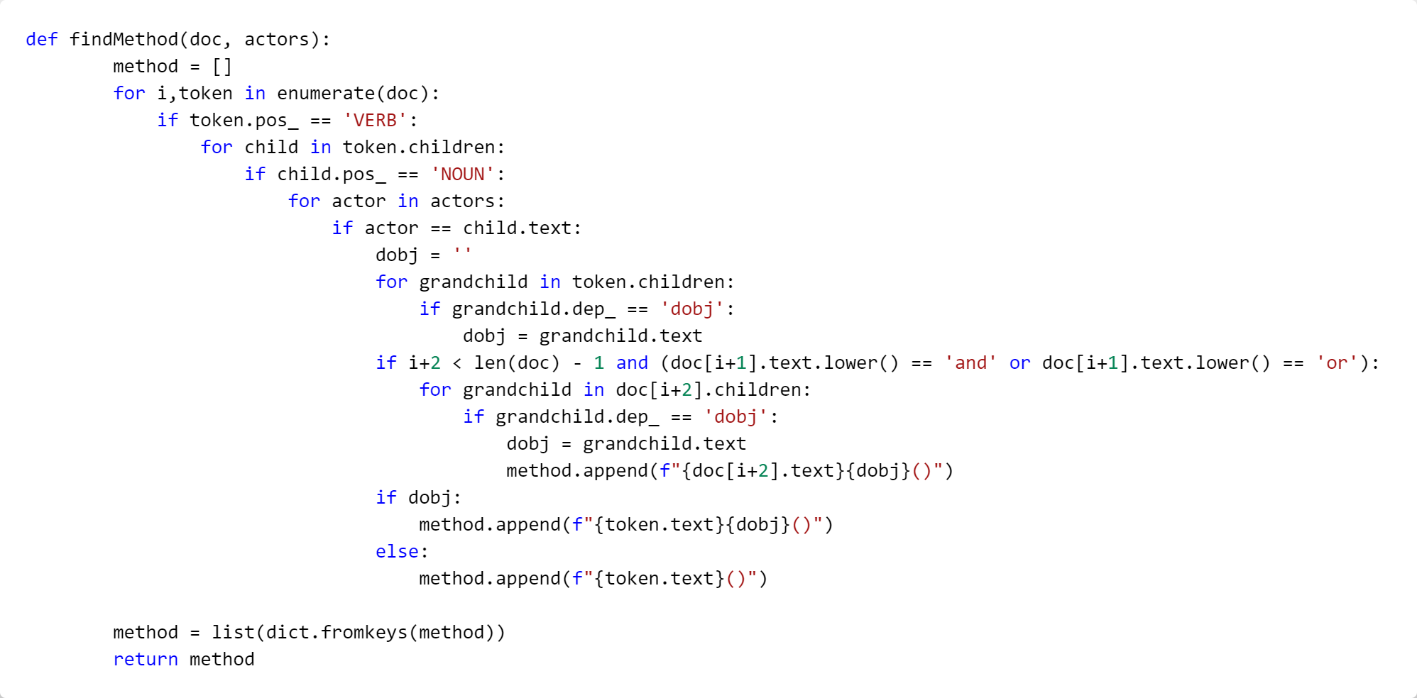


Figure 28: Showcases the findMethod(doc) method in our system.

This function, findMethod, is designed to find the methods in a given document (doc) based on the actors involved.

Here’s a step-by-step breakdown:

1. The function takes two arguments: a document (doc) and a list of actors (actors). The document is expected to be a processed text where each word is tokenized, and its part-of-speech (POS) is identified.
2. An empty list, method, is initialized to store the methods found in the document.
3. The function then iterates over each token in the document. For each token, it checks the following:
   * 1. If the token is a verb, it checks all its children. These are words that are related to this verb in the sentence structure.
     2. If any child of the verb is a noun and is also in the list of actors, the function begins to construct a method.
     3. The function initializes an empty string dobj to store the direct object of the verb (if any).
     4. The function checks the children of the verb again. If any child is a direct object (dobj), it stores the text of the child in dobj.
     5. If the current token and the next two tokens form a conjunction (i.e., the next token is ‘and’ or ‘or’), the function checks the children of the third token. If any child of the third token is a direct object, it stores the text of the child in dobj and appends the third token and dobj to the method list in the format of a method call (i.e., doc[i+2].textdobj()).
     6. If dobj is not empty, the function appends the verb and dobj to the method list in the format of a method call (i.e., token.textdobj()). Otherwise, it just appends the verb to the method list in the format of a method call with no arguments (i.e., token.text()).
4. After going through all the tokens, it removes duplicates from the method list by converting it to a dictionary and then back to a list.
5. Finally, it returns method, which is a list of methods found in the document according to the criteria defined above. Each method is a string that represents a method call performed by an actor. The method call consists of a verb (representing the action performed by the actor) and possibly a direct object (representing the object on which the action is performed).

A computer code with text

Description automatically generated

Figure 29: Showcases the findAttributes(doc) method in our system.

This function is designed to find the “attributes” in a given document (doc). Here, an “attribute” is defined as a noun or an adjective that is preceded by a pronoun or a particle.

Here’s a step-by-step breakdown:

1. The function takes a document (doc) as input. This document is expected to be a processed text where each word is tokenized and its part-of-speech (POS) is identified.
2. An empty list attributes is initialized to store the “attributes” found in the document.
3. The function then iterates over each token in the document. For each token, it checks the following:
4. If the token is a noun or an adjective, and the previous word is a pronoun or a particle, it adds the token to the attributes list.
5. If the word following the token is ‘and’, the word after it is also added to the attributes list.
6. Finally, attributes, a list of “attributes” found in the document according to the defined criteria, is returned.

## Testing

With the growing complexity in modern software applications, testing has become an integral part of the development process. Testing helps us to verify that the program works correctly and that it meets all the specified requirements. In the following sections of this report, we will review the main tests that we will conduct to ensure the quality of the program.

In the testing phase, we will do two main tests to make sure the program is good. The first is a system and scenario test. This is a type of Black Box Testing where we check that all parts of the program work together correctly and the program can handle different possible scenarios correctly. The second is a requirements test, also a type of Black Box Testing, where we check that all program requirements have been done correctly. It’s important to note that we did not do unit tests, which fall under White Box Testing, because we didn’t have enough time. This could be because of limits on resources or the project timeline. We will test the precision, recall, accuracy and F1 that were mentioned in the second chapter of the report.

### System testing

We will test 40 functional requirements. We will run the system and collect the outputs for each requirement. We will collect the actual outputs for each requirement. Once we have these outputs, we will evaluate the system's performance by calculating precision, recall, and precision metrics. These metrics will help us understand how the system performs in real situations and how close it is to achieving functional requirements. This testing will be necessary to ensure that the system is working properly according to the specified requirements.

1. **Identifying use case diagram element from requirement text**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Actual data** | | | | |
| Id | Requirements | actor | usecase | Actor to usecase |
| 1 | The user logs in to the system using their username and password. | user | login | user --> login |
| 2 | The admin updates the user's profile. | admin | Updates profile | admin --> updates profile |
| 3 | The system sends a notification to the user. | system | send notification | system --> send notification |
| 4 | The user can change their password at any time. | user | change password | user --> change password |
| 5 | The system automatically logs out after a period of inactivity for 15 minutes. | system | Auto logout | system --> Auto logout |
| 6 | The admin can add or delete users from the system. | admin | Manage user | admin --> Manage user |
| … | ……. | ......... | ............. | ....................... |
| 36 | The user can share their content with others. | user | share content | user --> share content |
| 37 | The system provides a search functionality with advanced filters. | system | Provide advanced search | system --> Provide advanced search |
| 38 | The admin can manage the system settings. | admin | manage system settings | admin --> manage system settings |
| 39 | The user can request support directly from the system. | user | request support | user --> request support |
| 40 | The system sends regular updates and patches. | system | sends updates and patches | system --> sends updates and patches |

Table 8: Actual data usecase elements

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Our system output** | | | | |
| Id | Requirements | actor | usecase | Actor to usecase |
| 1 | The user logs in to the system using their username and password. | user | logs | user --> logs |
| 2 | The admin updates the user's profile. | admin | Updates profile | admin --> updates profile |
| 3 | The system sends a notification to the user. | system | sends notification | system --> sends notification |
| 4 | The user can change their password at any time. | user | change password | user --> change password |
| 5 | The system automatically logs out after a period of inactivity for 15 minutes. | system | logs | system --> logs |
| 6 | The admin can add or delete users from the system. | admin | delete user,  add users | admin --> delete users, admin --> add users |
| … | ……. | ........... | ............. | ....................... |
| 36 | The user can share their content with others. | user | share content | user --> share content |
| 37 | The system provides a search functionality with advanced filters. | system | provides functionality | system --> provides functionality |
| 38 | The admin can manage the system settings. | admin | manage settings | admin --> manage settings |
| 39 | The user can request support directly from the system. | user, support | request support | user --> request support,  support --> request support |
| 40 | The system sends regular updates and patches. | system | sends updates | system --> sends updates |

Table 9: Our system output usecase

|  |  |
| --- | --- |
| Evaluation metric | Score |
| Precision | 0.67 |
| Recall | 0.85 |
| F1 score | 0.75 |
| accuracy | 0.7 |

Table 10: Evaluation metric result for usecase elements

1. **Identifying class diagram element from requirement text**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Actual data** | | | | |
| Id | Requirements | Class | Attributes | Method |
| 1 | The user logs in to the system using their username and password. | user | username, password | login() |
| 2 | The admin updates the user's profile. | admin | profile | updatesProfile() |
| 3 | The system sends a notification to the user. | system | Notification | sendsNotification() |
| 4 | The user can change their password at any time. | user | password | changePassword() |
| 5 | The system automatically logs out after a period of inactivity for 15 minutes. | system | lastActivityTime | autoLogout() |
| 6 | The admin can add or delete users from the system. | admin |  | deleteUser(), addUser() |
| … | ……. | ........... | ............. | ....................... |
| 36 | The user can share their content with others. | user | Content, sharedWith | shareContent() |
| 37 | The system provides a search functionality with advanced filters. | system | filters | search() |
| 38 | The admin can manage the system settings. | admin | settings | manageSettings() |
| 39 | The user can request support directly from the system. | user |  | requestSupport() |
| 40 | The system sends regular updates and patches. | system | Updates , patches | sendUpdates(),sendPatches () |

Table 11: Actual data class diagram elements

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Our system output** | | | | |
| Id | Requirements | Class | Attributes | Method |
| 1 | The user logs in to the system using their username and password. | user | username, password | logs() |
| 2 | The admin updates the user's profile. | admin | profile | updatesprofile() |
| 3 | The system sends a notification to the user. | system |  | sendsnotification() |
| 4 | The user can change their password at any time. | user | password | changepassword() |
| 5 | The system automatically logs out after a period of inactivity for 15 minutes. | system |  | logs() |
| 6 | The admin can add or delete users from the system. | admin |  | deleteusers(), addusers() |
| … | ……. | ........... | ............. | ....................... |
| 36 | The user can share their content with others. | user | content | sharecontent() |
| 37 | The system provides a search functionality with advanced filters. | system |  | providesfunctionality() |
| 38 | The admin can manage the system settings. | admin |  | managesettings() |
| 39 | The user can request support directly from the system. | user, support |  | requestsupport() |
| 40 | The system sends regular updates and patches. | system |  | sendsupdates() |

Table 12: Our system output class diagram elements

|  |  |
| --- | --- |
| Evaluation metric | Score |
| Precision | 0.65 |
| Recall | 0.89 |
| F1 score | 0.75 |
| accuracy | 0.68 |

Table 13: Evaluation metric result for usecase elements

|  |  |  |  |
| --- | --- | --- | --- |
| Description | Evaluation | Functional and non-Functional | Num |
|  |  | The system shall allow the user to enter requirements either by typing directly or by uploading a document. | 1 |
| Time constraints  big change in approach |  | The system shall allow the user to create analysis model | 2 |
|  |  | The system shall allow the user to choose type of UML diagram such as (class, use case). | 3 |
| Difficulty in programming  big change in approach |  | The system shall prepare to clean and organize the initial data to perform the text analysis process. | 4 |
| Difficulty in programming  big change in approach |  | The system should analyze the text by doing Tokenization, Parsing and Relation Extraction. | 5 |
|  |  | The system shall analyses the requirements context using NLP techniques. | 6 |
| Difficulty in programming  big change in approach |  | The system shall draw the diagram after generated by NLP techniques | 7 |
| Difficulty in programming  big change in approach  Time constraints |  | The system must be fast, ensuring that the processing of requirements and drawing (use case or class diagram) does not exceed (). | 8 |
| Difficulty in programming  big change in approach  Time constraints |  | The resulting diagrams in the system should have an accuracy of no less than (write the percentage). | 9 |
|  |  | It should be able to handle (write the range of users) simultaneously | 10 |
| Time constraints |  | The system should be user-friendly and adaptable to various devices to provide a seamless experience. | 11 |
|  |  | The system should perform its functions with minimal steps | 12 |
|  |  | The system should be able to recognize and analyze a minimum of (40%) of user requirements entered, with a permissible margin for error not exceeding (30%). | 13 |

### Functional testing

Table 14: Our system Functional testing

# Chapter 6: Conclusion and Future Work

## Conclusion

In this section, we will summarize all the work we have done from the beginning to the end of the project. We started by identifying the problem and how we would solve it, then defined the project's goals, methodology, and project plan.

In the second chapter, we researched topics related to the problem we are solving, including AI, ML, and NLP, and how to develop a system for artificial intelligence. We also explored similar systems.

In the third chapter, we analyzed existing systems, identified requirements from these systems, conducted brainstorming sessions, and then created the use-case diagram. We described the use cases and drew the class diagram.

In the fourth chapter we engaged in system design, where we outlined the architecture by creating a plan for building the system, identifying its main components and how they interact. We delved into component design, focusing on the details of each component's functions and interactions. Additionally, we developed the data model to organize and store data effectively. Lastly, we designed the user interface, defining screens and interactive elements for user interaction.

In the fifth chapter, we progressed to system development and testing. This involved presenting and explaining each method used in the development process. During testing, we conducted black-box testing, assessing the system against various use cases and requirements to ensure its functionality.

## Goals Achieved

We have successfully achieved our project goals. Here's a overview of the objectives and how we accomplished them.

1. **To review and delve deeper into related work in terms of design and code:**

A comprehensive review and in-depth exploration of relevant literature and existing work were conducted to enhance understanding of both design and code aspects.

1. **Improve the system analysis process:**

The system analysis process was enhanced and refined to ensure a more thorough and effective approach.

1. **Complete the design phase with a focus on the data model and components:**

The design phase was successfully finalized, with particular emphasis on refining the data model and components to meet project requirements.

1. **Program and test the system:**

System programming and rigorous testing procedures were carried out meticulously to ensure the system's functionality and reliability.

1. **Conclude with the lessons learned and the knowledge and experience gained from working on this project:**

We documented any new knowledge and experience gained from the beginning to the end of the project.

## Lessons Learnt

1. **Working as a team-** This project helped us to work together, learn and benefit from each other when we needed each other.
2. **Project planning**- involves setting the goals of the project, creating a task list for each goal, and developing a weekly schedule. Tasks are reviewed and assigned every weekend, and completed work is presented to the supervisor for feedback and approval.
3. **Learn NLP**- NLP is a new system for us, and we did not study anything about it in our university courses.
4. **Improve writing skills-** One of the most important things we learned in our project is to improve writing skill.
5. **Learning System Development for AI (NLP, ML)**- taught us the development steps in the fields of Natural Language Processing (NLP) and Machine Learning (ML).
6. **Researched and Read papers**- Acquired knowledge of methods for searching for reliable references and extracting information from them.
7. **Writing References**- Acquired knowledge of how to write references in the IEEE style.
8. **Every software project has different types of requirements**.

## Limitations and Future Work

### Limitations

The work we haven't done yet is reading user inputs and preparing them for analysis, as well as refraining from programming the front end.

### Future work

The tasks we will undertake in the second phase of the project include:

1. Enhance system development to recognize more patterns beyond functional requirements.
2. Conduct more extensive system testing with additional use cases.
3. Program user input reading and preprocessing for analysis.
4. Develop the front end.

# References

|  |  |
| --- | --- |
| [1] | H. Sofian, N. A. M. Yunus and R. Ahmad, "Systematic Mapping: Artificial Intelligence Techniques in Software Engineering," *IEEE Access,* vol. 10, pp. 51021-51040, 2022. |
| [2] | R. S. e. Bashir, "UML models consistency management: Guidelines for software quality manager.," *International Journal of Information Management,* vol. 36, no. 6, pp. 883-899, 2016. |
| [3] | T. w. l. t. p. o. f. t. a. o. technology, "The world's largest technical professional organization for the advancement of technology," IEEE. |
| [4] | university of BATH, 2017. [Online]. Available: https://www.bath.ac.uk/publications/library-guides-to-citing-referencing/attachments/ieee-style-guide.pdf. |
| [5] | ". R. M. A. D. Library. [Online]. Available: https://dl.acm.org/. |
| [6] | G. F. Luger., Artificial intelligence: structures and strategies for complex problem solving., Addison-Wesley Publishing CompanyUnited States, 2008. |
| [7] | 2. W. A. Hany H Ammar1, Software Engineering Using Artificial Intelligence Techniques: Current State and Open Problems, 2013. |
| [8] | S. Shafiq, A. Mashkoor, C. Mayr-Dorn and A. Egyed, "A Literature Review of Using Machine Learning in Software Development Life Cycle Stages," *IEEE Access,* vol. 9, pp. 140896-140920, 2021. |
| [9] | Y. Vasiliev, Natural language processing with Python and spaCy: A practical introduction., No Starch Press , 2020. |
| [10] | E. &. C. M. B. Dias Canedo, " Software requirements classification using machine learning algorithms.," *Entropy,* vol. 22, no. 9, p. 1057, 2020. |
| [11] | S. Vajjala, B. Majumder, A. Gupta and H. Surana, "Practical Natural Language Processing: A Comprehensive Guide to Building Real-world NLP Systems.," O'Reilly Media, 2020. |
| [12] | H. a. A. W. B. Herchi, "From user requirements to UML class diagram," *arXiv preprint arXiv:1211.0713,* 2012. |
| [13] | P. P.-S. Chen, "English sentence structure and entity-relationship diagrams," *Information Sciences,* vol. 29, pp. 127-149, 1983. |
| [14] | K. A. D. O. K. K. Arachchi, AI Based UML Diagrams Generator, 2021. |
| [15] | S. Ian, Software Engineering, 11th ed., Addison-Wesley,, 2015. |
| [16] | R. J. Leach, Introduction to Software Engineering, 2016. |
| [17] | B. B. a. D. A. H.., Object-oriented Software Engineering: Using UML, Patterns and Java, 2009. |
| [18] | G. L. S. C. &. H. O. S. Moon, "Automated construction specification review with named entity recognition using natural language processing," *Journal of Construction Engineering and Management,* vol. 147, no. 1, p. 04020147, 2021. |
| [19] | S. &. C. C. Jaf, "Deep learning for natural language parsing," *IEEE Access,* vol. 7, pp. 131363-131373, 2019. |
| [20] | R. P. S. C. E. &. T. R. Sukthanker, "Anaphora and coreference resolution: A review," *Information Fusion,,* vol. 59, pp. 139-162, 2020. |
| [21] | C. Manning, "Stanford lecture (CS224n) by Christopher Manning," 2019. [Online]. Available: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1162/handouts/cs224n-lecture10-coreference.pdf. |
| [22] | S. G. L. &. H. C. Loáiciga, "What is it? Disambiguating the different readings of the pronoun 'it.," *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing,* pp. 1325-1331, 2017. |
| [23] | H. a. Y. N. A. M. a. A. R. Sofian, "Systematic mapping: Artificial intelligence techniques in software engineering," *IEEE Access,* vol. 10, pp. 51021-51040, 2022. |
| [24] | Z. a. L. F. a. Y. W. a. P. S. a. Z. J. Li, "A survey of convolutional neural networks: analysis, applications, and prospects," *IEEE transactions on neural networks and learning systems,* 2021. |
| [25] | V. Nasteski, "An overview of the supervised machine learning methods," *Horizons. b,* vol. 4, pp. 51-62, 2017. |
| [26] | J. T. Catanio, "Requirements analysis: A review," *Advances in Systems, Computing Sciences and Software Engineering,* pp. 411-418, 2006. |
| [27] | B. Arendse, "A thorough comparison of NLP tools for requirements quality improvement," 2016. |
| [28] | M. O. I. Bashir, "ResearchGate," [Online]. Available: https://www.researchgate.net/figure/Different-branches-of-AI-13\_fig1\_357512563. |
| [29] | JayeshBapuAhire, "Data Science Central," [Online]. Available: https://www.datasciencecentral.com/the-artificial-neural-networks-handbook-part-1/. |
| [30] | P. M. Marta Maślankowska, "neurosys," [Online]. Available: https://neurosys.com/blog/intro-to-coreference-resolution-in-nlp. |
| [31] | "UML/Code Generation Tool," visual paradigm, [Online]. Available: https://www.visual-paradigm.com/features/code-engineering-tools/. |
| [32] | "Defining design pattern," visual paradigm, [Online]. Available: https://www.visual-paradigm.com/support/documents/vpuserguide/26/36/6246\_definingdesi.html. |
| [33] | "lucid collaborative ai," innovation training, [Online]. Available: https://www.innovationtraining.org/lucid-collaborative-ai/. |
| [34] | "lucidchart vs lucidspark," innovation training, [Online]. Available: https://www.innovationtraining.org/lucidchart-vs-lucidspark-when-why-and-how-to-use-them-both/. |
| [35] | "starUML in 2022," Pat research, [Online]. Available: https://www.predictiveanalyticstoday.com/staruml/#content-anchor. |
| [36] | "Introduction to Enterprise Architect," sparx systems, [Online]. Available: https://sparxsystems.com/enterprise\_architect\_user\_guide/13.0/. |
| [37] | "MagicDraw," Dassault Systemes, [Online]. Available: https://www.3ds.com/products-services/catia/products/no-magic/magicdraw/. |
| [38] | B. B. a. D. A. H, Object-oriented Software Engineering: Using UML, Patterns and Java, 2009. |