Title of the Final Year Undergraduate Project

An undergraduate project proposal report submitted to the

Department of Electrical and Information Engineering
Faculty of Engineering
University of Ruhuna
Sri Lanka

in partial fulfillment of the requirements for the

Degree of the Bachelor of the Science of Engineering Honours

by

R. Lareef	-	EG/2021/4644
A. Abekon	-	EG/2021/4645
B. Safeer	-	EG/2021/4771

....

Prof. A.B.C. Dee
(Supervisor)

Prof. A.B.C. Dee
(Co-Supervisor)

Abstract

The growing complexity of modern software systems and the need for effective communication among stakeholders have made Unified Modeling Language (UML) diagrams a vital component of software engineering. However, manually creating UML diagrams from textual requirements is a time-consuming, error-prone process that requires considerable expertise. To address this challenge, this research proposes an AI-driven framework for automatically generating UML class and use case diagrams from natural language software requirements.

The proposed system leverages advanced Natural Language Processing (NLP) and Deep Learning (DL) techniques to interpret unrestricted, ambiguous, and incomplete textual inputs more accurately. By improving grammatical and semantic understanding, the framework aims to produce more reliable and comprehensive UML representations.

The methodology includes both quantitative and qualitative evaluation. Accuracy is measured using standard metrics such as precision, recall, and F1-score, while usability is assessed through feedback from students and industry professionals. A comparative analysis with existing tools demonstrates that the proposed approach offers improved scalability, robustness, and efficiency.

This research contributes to the field of software engineering by reducing manual effort, enhancing design accuracy, and improving collaboration among stakeholders. By automating UML diagram generation, the proposed framework supports more efficient and adaptive software development workflows, benefiting both academic research and industrial practice.

Contents

Acronyms **vi**

1 Introduction **1**
1.1 Background 1
1.2 Problem Statement 2
1.3 Objectives and Scope 2

2 Literature Review **3**
2.1 Previous Work 3
2.2 Gaps in Literature 4

3 Methodology **6**
3.1 Research Design 6
3.2 Data Collection 6

4 Timeline and Resource Required **7**
4.1 Timeline 7
4.2 Resource Required 7

5 Conclusion **8**

References **8**

List of Figures

3.1 This is an example image. 6

List of Tables

Acronyms

AI	Artificial Intelligence
ML	Machine Learning

Chapter 1

Introduction

1.1 Background

The rapid growth of technology and the expanding influence of the software and IT industry have significantly transformed how individuals and organizations operate. Software applications now play a central role in driving innovation across domains such as healthcare, education, finance, and entertainment. As software systems become more complex, the demand for efficient, accurate, and collaborative software development processes has increased. Effective communication tools and clear design methodologies are essential to ensure a shared understanding among all stakeholders involved in software projects.

Unified Modeling Language (UML) diagrams are widely used to represent software designs in a standardized and visual manner. They help illustrate system components, interactions, and workflows, making them understandable to both technical and non-technical users. However, creating UML diagrams manually from textual software requirements is a time-consuming and expertise-dependent task. This process often leads to inconsistencies, misinterpretations, and inefficiencies, especially in large-scale projects where speed and accuracy are critical.

Recent advancements in Artificial Intelligence (AI), particularly in Natural Language Processing (NLP) and Deep Learning (DL), provide a promising solution to this problem. NLP enables machines to understand and analyze human language, while DL models can learn complex patterns and generate meaningful outputs. By leveraging these technologies, our project aims to automate the generation of UML diagrams—specifically class diagrams and use case diagrams—from natural language software requirements. This approach seeks to provide a more comprehensive and practical solution for software design automation.

Existing automated systems face several limitations, including the inability to handle incomplete requirement documents, restrictions on sentence structure, and difficulties in understanding grammatical nuances. These issues often result in inaccurate or incomplete UML diagrams. Our proposed system addresses these challenges by supporting unrestricted natural language input, improving grammatical analysis, and handling missing information more effectively.

From a practical and commercial perspective, automating UML diagram generation can significantly reduce development time and cost. It can improve team productivity, enhance collaboration among distributed teams, and ensure consistency in design documentation. By improving efficiency, accuracy, and scalability, our solution can help organizations gain a competitive advantage in today's fast-paced software industry.

This project proposes a robust AI-driven framework that converts textual software requirements into accurate UML diagrams. By bridging the gap between natural language and visual system models, our work contributes to the integration of AI technologies into the software development lifecycle.

1.2 Problem Statement

Despite the significant role that UML diagrams play in software development, existing tools for generating them from natural language requirements have several limitations. Most current solutions focus primarily on class diagrams, while neglecting other important diagram types such as use case diagrams. Furthermore, these tools struggle with incomplete requirement documents, restricted input formats, and grammatical inconsistencies, which often result in errors and inefficiencies. As a result, considerable manual intervention is required, reducing the scalability, accuracy, and practicality of these systems in real-world software development environments.

1.3 Objectives and Scope

State the objectives of your project clearly.

Chapter 2

Literature Review

2.1 Previous Work

The extraction of UML class diagrams from informal software requirements has traditionally been addressed using Natural Language Processing (NLP) techniques combined with heuristic rules or domain ontologies. Over time, researchers have explored more advanced approaches, including machine learning (ML) and deep learning (DL), to improve automation, accuracy, and efficiency. Existing studies mainly fall into three categories: NLP-based, ML-based, and DL-based approaches [?].

NLP-Based Approaches

Ibrahim and Ahmad introduced the RACE tool, which uses NLP techniques to analyze software requirements and extract UML class diagrams. The process includes structural analysis, refinement of diagram elements, and validation of UML concepts. However, the generated diagrams often lack complete information [?].

Kumar and Sanyal developed the SUGAR tool using the Rational Unified Process (RUP) methodology. It generates UML class diagrams from natural language requirements by first creating use case diagrams and then linking them to class diagram components. This approach supports object-oriented analysis but still relies heavily on predefined structures [?].

Sharma et al. proposed the Functional Design Creation Tool (FDCT), which preprocesses and analyzes requirements to identify model entities and integrate them into high-level class diagrams [?].

Deeptimahanti and Sanyal introduced UMGAR, a semi-automatic system that uses syntactic rules and analysis trees to extract UML elements. Although it can generate multiple UML diagram types, it requires human intervention to refine relationships and remove irrelevant elements [?].

Similarly, the RAPID tool by More and Phalnikar can generate use case, sequence, and collaboration diagrams using an architecture similar to RACE [?].

Shinde et al. developed a system that applies NLP preprocessing and rule-based extraction to generate UML class and sequence diagrams, and even produce Java source code from the extracted models [?].

The UDRA tool by Amune and Naik combines NLP with domain ontologies to generate various UML diagrams such as class, object, package, and deployment diagrams. It supports relationships like association, generalization, and composition [?].

Other language-specific approaches include the work of Jaiwai and Sammapun, who developed a system for Thai-language requirements, and Utama and Jang, who used SpaCy with linguistic rules to extract UML class diagrams from unrestricted natural language [?].

Abdelnabi et al. proposed an NLP and heuristic-based method for generating class diagrams from informal requirements [?].

Bashir et al. developed the READ tool, which uses the NLTK library and rule-based extraction to automatically identify classes, attributes, and operations. Their results showed improved accuracy compared to earlier tools [? ?].

Machine Learning and Deep Learning Approaches

Although NLP-based systems improved automation, they still struggle with ambiguity, incompleteness, and uncertainty in natural language requirements [?].

Arachchi proposed a hybrid system that combines NLP, machine learning, and deep learning to generate UML class and use case diagrams. The system uses NLP for text processing, Naive Bayes for classification, and RNNs for identifying actors, classes, and relationships [?].

Saini et al. introduced DoMoBOT, which integrates NLP with ML and DL models to generate domain models as UML class diagrams. The system preprocesses text using SpaCy, extracts domain concepts using NLP rules, refines them using trained ML/DL models, and integrates the results into a final UML model [?].

Rigou and Khriss proposed a deep learning-based framework using Bi-LSTM, BERT, and GPT-2 models to extract UML class diagrams from natural language requirements. Their system identifies entities, removes redundant references, and classifies relationships using neural networks. Results showed that larger training datasets improved diagram accuracy [? ?].

2.2 Gaps in Literature

Despite significant progress, several limitations remain in existing UML diagram extraction systems:

- **Limited Diagram Coverage:** Most tools primarily focus on generating UML class diagrams, while other important diagram types such as use case diagrams, sequence diagrams, and activity diagrams receive less attention.
- **Dependence on Rule-Based Systems:** Many approaches rely on fixed linguistic rules and heuristics, making them less flexible when dealing with diverse writing styles, complex sentence structures, or informal requirements.
- **Handling of Incomplete Requirements:** Existing systems struggle to generate accurate diagrams when requirement documents are missing key information, leading to incomplete or incorrect UML models.

- **Grammatical and Semantic Challenges:** Ambiguity, grammatical inconsistencies, and contextual misunderstandings in natural language often cause extraction errors, even in advanced NLP-based tools.
- **High Manual Intervention:** Several systems require human input to refine relationships, remove irrelevant elements, or correct extracted models, reducing scalability and automation.
- **Limited Real-World Scalability:** Many tools are tested only on small datasets or controlled inputs, making their performance in large-scale, real-world projects uncertain.
- **Restricted Input Formats:** Some systems require standardized or structured requirements, limiting their usability with freely written natural language documents.

Chapter 3

Methodology

3.1 Research Design

Describe the research design and methodology. Use diagrams or flow charts to illustatre the methodology.

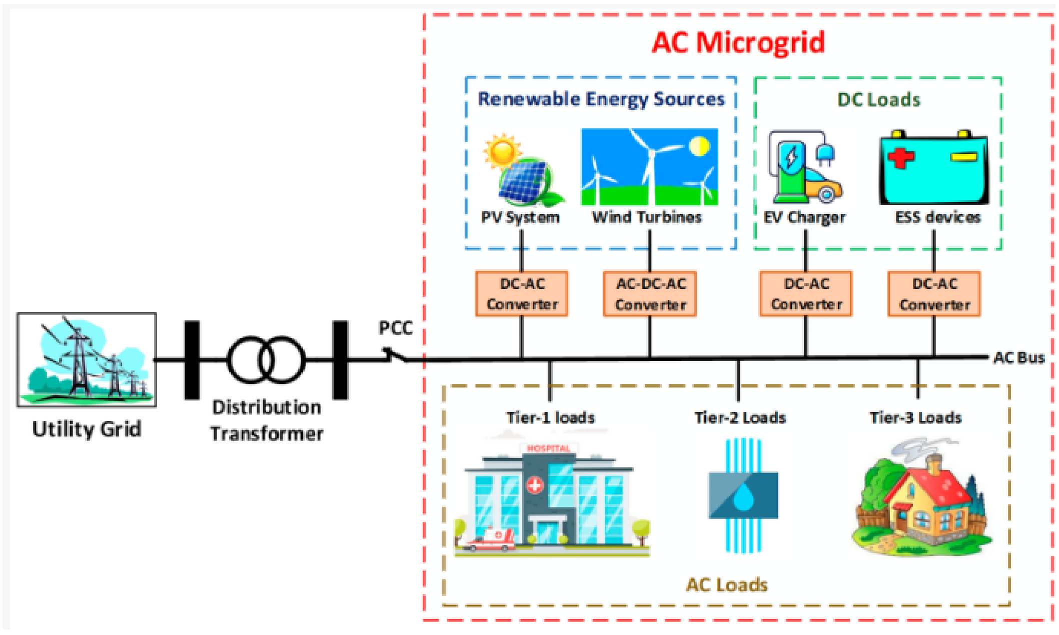

Figure 3.1: This is an example image.

As shown in Figure 3.1, the image is centered and has a caption.

3.2 Data Collection

Explain how data will be collected for the project.

Chapter 4

Timeline and Resource Required

4.1 Timeline

Provide a realistic timeline for completing the project. Use a Gantt chart or table.

4.2 Resource Required

State the resource required for the project and estimate the budget for the resources required.

Chapter 5

Conclusion

Summarize the key points of your proposal and reiterate the importance of the project.

References

[1] A. Jones, *Machine Learning: Theory and Practice.* Tech Publications, 2019.