Faculty of Computer Science and Information Technology

Visualisation of Textual User Stories to UML Use Case Diagram: A Natural Language Processing Approach

Mohammad Nazrul bin Mornie

Master of Science
2026

Visualisation of Textual User Stories to UML Use Case Diagram: A Natural Language Processing Approach

Mohammad Nazrul bin Mornie

A thesis submitted

In fulfillment of the requirements for the degree of Master of Science

(Language Technology)

Faculty of Computer Science and Information Technology
UNIVERSITI MALAYSIA SARAWAK
2026

DECLARATION

I declare that the work in this thesis was carried out in accordance with the regulations of Universiti Malaysia Sarawak. Except where due acknowledgements have been made, the work is that of the author alone. The thesis has not been accepted for any degree and is not concurrently submitted in candidature of any other degree.

…….

Signature

Name: Mohammad Nazrul bin Mornie

Matric No.: 21020048

Faculty of Computer Science and Information Technology

Universiti Malaysia Sarawak

ACKNOWLEDGEMENT

I want to take this opportunity to express my sincere gratitude to all individuals and institutions who have contributed directly or indirectly to the successful completion of this thesis.

First and foremost, my deepest and most special appreciation goes to my main supervisor, Ts. Nurfauza Binti Jali, for her continuous guidance, patience, encouragement, and invaluable support throughout this research journey. Without her dedication and constructive guidance, the completion of this thesis would have been extremely challenging.

I would also like to extend my sincere appreciation to the examiners for their time, constructive comments, and valuable feedback, which have significantly enhanced the quality and depth of this thesis.

My heartfelt thanks go to my mother and family for their unwavering support, encouragement, and understanding throughout my studies. I am also grateful to all lecturers who were involved in this journey for their assistance, guidance, and insightful advice. Without their support, the completion of this research and the publication of my paper would not have been possible.

Furthermore, I would like to express my gratitude to Universiti Malaysia Sarawak (UNIMAS) for sponsoring my studies under the Biasiswa UNIMAS Prihatin 2021, which provided essential financial support throughout my academic journey. Finally, my sincere appreciation goes to the Centre for Graduate Studies (CGS), Universiti Malaysia Sarawak, for the continuous advice and support provided during my period of study.

ABSTRACT

The increasing adoption of Agile methodology in software development by both industry professionals, such as software engineers, system analysts, and requirements engineers, including academia students has highlighted the importance of using UML diagrams for requirements modelling. Among these, use case diagrams are especially useful for capturing user requirements and helping development teams understand system functionalities and user interactions. However, there is currently a lack of tools capable of directly generating use case diagrams from textual user stories. Creating these diagrams manually requires a strong understanding of the requirements, effective communication with stakeholders, and is often time-consuming. Moreover, previous studies have not fully addressed the accurate representation of relationship elements in use case diagrams. This study proposes a method to visualise use case diagrams from structured textual user stories by applying Natural Language Processing (NLP) techniques and logical rules. The approach is carried out in four stages: Requirements gathering, Natural Language Processing, Application of Logical Rules, and UML Diagram Generation. The Stanza, which is a Python interface of Stanford CoreNLP toolkit is employed to perform tokenisation, stemming and lemmatisation, part-of-speech tagging, and dependency parsing. Logical rules are then applied to generate the final diagram. This method aims to bridge the existing gap in representing relationships accurately and introduces a semi-automated approach for generating use case diagrams from textual user stories.

Keywords: Requirement Engineering, Textual user stories, Use Case Diagram, UML Model, Natural Language Processing

Visualisasi Cerita Pengguna Berbentuk Teks kepada Rajah Kes Penggunaan UML: Pendekatan Pemprosesan Bahasa Semula Jadi

ABSTRAK

Peningkatan penggunaan metodologi Agile dalam projek pembangunan perisian oleh para profesional seperti jurutera perisian, penganalisis sistem, dan jurutera keperluan, termasuk pelajar dalam bidang akademik, telah menekankan kepentingan penggunaan rajah UML untuk pemodelan cerita pengguna. Antara rajah UML tersebut, rajah kes kebergunaan amat berguna untuk menangkap keperluan pengguna serta membantu pasukan pembangunan memahami fungsi sistem dan interaksi antara pengguna dan sistem. Namun begitu, masih tiada alat yang mampu menjana rajah kes kebergunaan secara langsung daripada cerita pengguna. Proses pembinaan rajah ini secara manual memerlukan pemahaman yang mendalam terhadap keperluan, komunikasi yang berkesan dengan pihak berkepentingan, dan sering kali mengambil masa yang lama. Selain itu, kajian terdahulu juga belum dapat menghasilkan unsur hubungan dalam rajah kes kebergunaan dengan tepat. Kajian ini mencadangkan satu kaedah untuk menjana rajah kes kebergunaan daripada cerita pengguna berstruktur berbentuk teks dengan menggunakan teknik Pemprosesan Bahasa Semula Jadi dan penerapan peraturan logik. Pendekatan ini dijalankan melalui empat peringkat iaitu Pengumpulan Keperluan, Pemprosesan Bahasa Semula Jadi, Penerapan Peraturan Logik, dan Penjanaan Rajah UML. Stanza merupakan antara muka Python kepada Stanford CoreNLP digunakan untuk melaksanakan empat teknik NLP iaitu tokenisasi, stemming dan lemmatisasi, penandaan bahagian pertuturan, dan penghuraian pergantungan. Seterusnya, peraturan logik digunakan untuk menjana rajah akhir. Kaedah ini bertujuan untuk menangani jurang sedia ada dalam perwakilan unsur hubungan dan memperkenalkan pendekatan separa automatik bagi penjanaan rajah kes kebergunaan daripada cerita pengguna.

Kata kunci: *Kejuruteraan Keperluan, Cerita Pengguna, Rajah Kes Kebergunaan, Rajah UML, Pemprosesan Bahasa Semula Jadi*

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

UNIMAS	Universiti Malaysia Sarawak
UML	Unified Modelling Language
NLP	Natural Language Processing
US2UCD	Textual user stories to UML Use Case Diagram
SKU	Stock Keeping Unit
XMI	XML Metadata Interchange
ERD	Entity-Relationship-Diagram
AI	Artificial Intelligence

CHAPTER 1

INTRODUCTION

1.1 Study Background

Unified Modelling Language (UML) is a modelling language used during the design phase of the Agile methodology. UML consists of different types of diagrams that can help the developer during the development of software. The most obvious benefit of using UML in Agile methodology is that it can help software developers visualise and model complex and relatively large systems. Different types of UML models used in Agile methodology include use case diagrams, class diagrams, sequence diagrams, and activity diagrams. On the other hand, textual user stories also play a significant role in Agile methodology, as they help developers understand software requirements from an end-user perspective. This is because textual user stories help developers get ideas about the software from the end user's perspective. Textual user stories are also crucial for specifying software requirements. Once the software requirements have been specified, the need arises to design the software, including preparing UML models such as use case diagrams, activity diagrams, and sequence diagrams. This project proposes a framework to automatically generate UML use case diagrams from textual user stories using Natural Language Processing (NLP).

1.2 Problem Statement

The rapid advancement of Information Technology has increased the demand for automated and intelligent approaches to improve the efficiency of software development processes. In Agile methodology, textual user stories are widely used to capture functional requirements in a natural language format. At the same time, UML diagrams are used during the design phase to formally represent system structure and behaviour. However,

transforming textual user stories into UML models remains a complex and intensive task that relies heavily on manual interpretation by developers. From a computer science perspective, this challenge arises due to the unstructured and ambiguous nature of natural language in textual user stories. This makes it difficult to systematically extract actors, actions, relationships, and system components required for UML modelling. The manual translation process is time-consuming, error-prone, and inconsistent, especially in large-scale Agile projects. This limitation directly impacts development speed, design accuracy, and overall software quality. Despite advances in NLP and automated software engineering, existing approaches do not sufficiently address the problem of accurately mapping user story semantics into formal UML representations in an automated manner. Therefore, there is a need for a computational approach that applies NLP techniques to analyse textual user stories and automatically generate UML models. By having this, there will be improved consistency, reduced human effort, and enhanced productivity during the Agile design phase.

1.3 Research Questions

As the problems are discussed, a few research questions have been identified. The questions are critical because it will play a huge role in completing this research. The research questions that have been identified are:

i. What are the current limitations of automating UML models generation through textual user stories?
ii. How can the generation of UML models from textual user stories be automated?
iii. How to evaluate the effectiveness of the framework for automating UML models generations from textual user stories?

1.4 Research Aim and Objectives

The aim of this research is to introduce a framework for automating the generation of use case diagrams from textual user stories. By having this framework, it will be much more efficient for an agile software development team to document the requirements of the software. Based on the research aim, a set of objectives is formulated to achieve it. The objectives of the study are as follows:

i. To study the current limitations of automating UML models generation through textual user stories.
ii. To design a framework for automating the generation of a UML use case diagram from textual user stories using the natural language processing approach.
iii. To evaluate the framework using a developed prototype.

1.5 Research Scope

This research aims to propose a framework for generating a UML use case diagram from textual user stories. A prototype will be developed as part of the study, targeting users in academia, including software engineering students, as well as industry professionals such as software engineers, requirements engineers, and software analysts. The scope of this research is outlined below for clarity.

i. A prototype is developed to target users in academia, specifically Software Engineering students from the Faculty of Computer Science and Information Technology, as well as industry professionals such as software engineers, requirements engineers, and system analysts.
ii. The prototype is designed for personal computers (PCs).
iii. The research focuses on a single UML model which is the automatic generation of use case diagrams from textual user stories.

iv. The proposed framework in this research handles only textual user stories written in English.

1.6 Research Significance

The completion of this research is expected to benefit both academic research in computer science and industry practitioners, such as software engineers and system analysts. From a Software Engineering perspective, this research contributes by proposing a structured framework for automatically generating UML use case diagrams from textual user stories. This approach introduces an alternative way of documenting software requirements, particularly in Agile development, where user stories are commonly used but design models are still produced manually.

In current practice, use case diagrams are usually created by manually drawing them or using online diagramming tools, which can be time-consuming and may lead to inconsistent results depending on the individual's interpretation. The framework proposed in this research applies a defined algorithm to analyse textual user stories before generating the use case diagram, allowing the transformation process to be carried out in a more systematic manner. The effectiveness of the framework is evaluated using precision, recall, F1-score, and completeness by comparing the generated UML use case diagrams with manually created reference diagrams.

Overall, this research contributes to the body of knowledge in Software Engineering by demonstrating how computational techniques can be applied to automate the transformation of informal requirements into formal use case diagrams. The findings of this study can also serve as a reference for future research in automated requirements modelling and UML diagram generation.

CHAPTER 2

BACKGROUND AND LITERATURE REVIEW

2.1 Review Background

In this chapter, the background and related literature are explored, with a focus on the generation of UML models from textual user stories using Natural Language Processing (NLP). While the primary concern is UML model generation, studies involving other conceptual models such as Entity Relationship Diagrams (ERD) or business process models are also considered, provided they employ NLP techniques. The main components of this domain are UML, textual user stories, and NLP.

Agile software development, widely adopted across the industry, emphasizes iterative development, stakeholder collaboration, and adaptability to requirement changes (Thesing et al., 2021). Within Agile, textual user stories serve as a common format to capture functional requirements from the user's perspective (Cohn, 2015; Elallaoui et al., 2018). The user stories also describe the users, their goals, and the reasons behind those goals (Bik et al., 2017), helping stakeholders and developers align on software functionality. However, due to their informal nature, textual user stories can lead to ambiguity, overlooked non-functional requirements, and prioritization issues if not properly written (Gilson & Irwin, 2018).

To complement textual user stories, UML (Unified Modeling Language) offers a visual representation of software systems, making it easier for developers to interpret requirements. Common UML diagrams include use case, class, sequence, activity, and state diagrams (Koç et al., 2021; Seidl et al., 2015). Among the different types of UML diagrams, use case diagrams hold particular significance in Agile software development. This is

because they provide a high-level visual representation of actors, their interactions with the system, and the expected system behaviour, which aligns closely with the user-centric philosophy of Agile. Unlike detailed design models, use case diagrams offer quick, intuitive insights into system functionality without requiring extensive technical knowledge, making them accessible to both developers and non-technical stakeholders. As such, they serve as an effective communication bridge, allowing stakeholders to validate requirements early and ensure alignment with business objectives. Additionally, use case diagrams support more structured documentation of requirements, which is crucial in Agile contexts where requirements evolve rapidly, and informal user stories can lead to ambiguities. By providing a formalised yet easily understandable representation, use case diagrams help reduce misunderstandings, facilitate prioritising features, and guide subsequent modelling tasks, such as generating sequence or class diagrams. The automation or semi-automation of use case diagram generation using NLP tools and techniques further enhances efficiency and ensures that even large volumes of user stories can be systematically transformed into structured models without losing critical information.

Natural Language Processing (NLP) plays a central role in this research because the focus is on textual user stories, which are written in natural, human language. User stories are widely used in Agile development to capture functional requirements in a simple, accessible format for both technical and non-technical stakeholders. However, their informal nature can lead to ambiguity, inconsistency, and incomplete information, making it difficult to produce UML models directly. NLP enables systematic processing of unstructured text to extract structured information, such as actors, use cases, classes, and relationships, which can then be mapped to UML or other conceptual models (Zhao et al., 2021). Techniques such as tokenisation, part-of-speech tagging, lemmatisation, and dependency parsing allow these

informal descriptions to be transformed into formal representations, reducing human interpretation errors and supporting more accurate model generation (Alzayed and Al-Hunaiyyan, 2021). Various NLP tools, including Stanford CoreNLP, spaCy, NLTK, and OpenNLP offer the linguistic and syntactic analysis capabilities necessary to interpret natural language requirements systematically. By using NLP, this research justifies the reliance on natural language as it enables the automated or semi-automated transformation of user stories into structured UML models, capturing the richness of real-world requirements while ensuring consistency, reproducibility, and scalability.

2.2 Review Method

This research adopted a systematic literature review to investigate the generation of UML models from textual user stories using Natural Language Processing. The review followed a structured process that included planning and executing the search, applying selection criteria, and synthesising the findings. This approach was chosen to ensure a comprehensive and objective analysis of relevant studies. The review was guided by three explicit research questions which are:

i. **RQ1** – What are the challenges faced by academics and the industry in designing UML models?
ii. **RQ2** – What are the NLP tools used to assist the generation of UML models from user stories?
iii. **RQ3** – How to measure the accuracy of UML models or conceptual models generated from user stories?

To gather the literature, four academic databases, which are Scopus, ScienceDirect, IEEE Xplore, and the ACM Digital Library were selected based on their coverage of high-

quality publications in software engineering and computing. Limiting the search to these databases ensured relevance and accessibility, although it is recognised that this may have excluded some studies published in other repositories or journals. Each database was searched using a consistent set of terms related to the research topic. Keywords included "Natural Language Processing", "UML model", "textual user stories", and "generate", along with alternative expressions such as "transform", "analyse", and "visualise". Boolean operators were used to refine and broaden the search where appropriate. The search was limited to journal articles and conference papers published between 2018 and 2022, and only English-language publications were considered to maintain consistency and quality.

Inclusion and exclusion criteria were applied to select the most relevant studies. Papers were included if they addressed the use of NLP in generating UML or other conceptual models, discussed the challenges involved in the process, described the tools or techniques used, or presented validation of the models produced. Review articles were excluded because the focus of this research is on primary empirical and methodological studies that directly investigate UML generation from textual user stories, rather than summarising previous reviews. Studies were also excluded if they were published in languages other than English, published before 2017, or categorised as lecture notes or research proposals. Duplicate records across databases were removed. In total, 198 studies were retrieved. After applying the criteria and conducting full-text assessments, 20 primary studies were selected for detailed analysis. While the inclusion criteria were designed to capture the most relevant studies, it is acknowledged that the criteria could be further refined to incorporate aspects such as study quality or citation impact, which may have strengthened the selection process. These 20 primary studies form the basis for the findings and discussion presented in the following sections.

2.3 Review Results

Following the application of inclusion and exclusion criteria, a total of 20 primary studies were selected for this review. These studies were published between 2018 and 2025 and sourced from four databases which are Scopus, ScienceDirect, IEEE Xplore, and ACM Digital Library. Scopus provided the largest number of relevant papers. The selection process involved initial screening, removal of duplicates, abstract review, and full-text evaluation to ensure the relevance and quality of each study. A keyword network visualisation shown in Figure 2.1 was created using VOSviewer to identify patterns and relationships among key terms used by researchers in the selected studies.

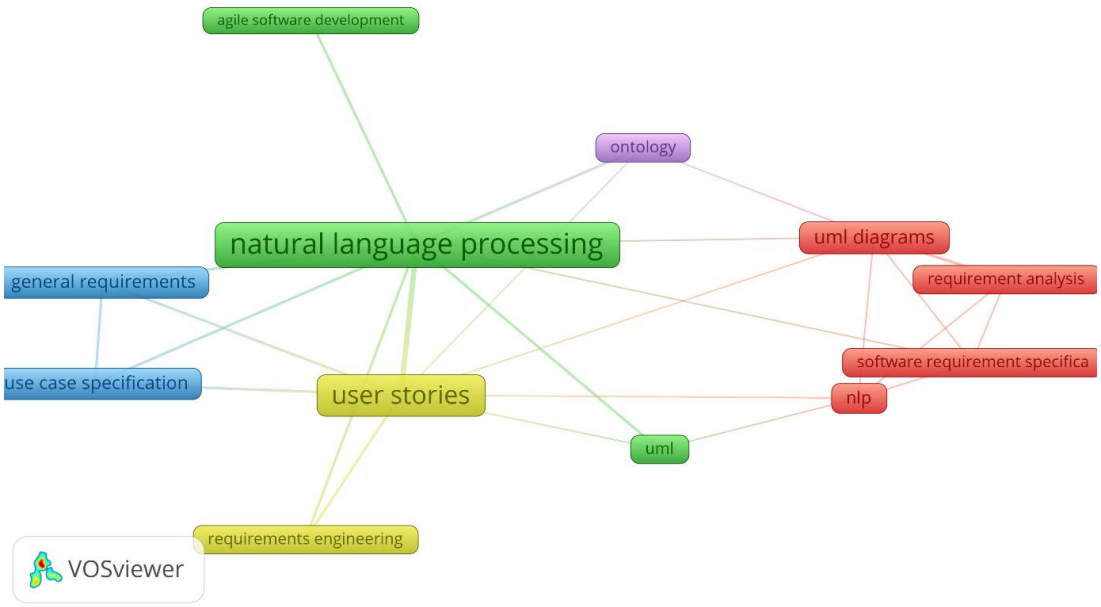


Figure 2.1: Visualisation of keywords used from the selected primary studies

2.3.1 Challenges faced by academics and industry professionals in designing the UML model or another conceptual model

In Agile software development, UML and other conceptual models are typically created during the design and analysis phases. However, this process presents several challenges, particularly when performed manually. Multiple studies reported that generating UML diagrams, such as sequence or activity diagrams, requires considerable time and effort, and may incur high costs (Abdelnabi et al., 2021; Alami et al., 2020; Fischbach et al., 2020; Maatuk & Abdelnabi, 2021). These challenges affect both industry professionals and students in academic settings.

A major difficulty comes directly from the nature of natural language. Textual user stories and requirements are inherently ambiguous, redundant, incomplete, or inconsistent. Such characteristics make it challenging to accurately extract key elements, such as actors, use cases, and their interactions (Vasques et al., 2019; Ternes et al., 2021). Reliance on human interpretation increases the risk of errors, especially when analysts must review a large volume of user stories, which may lead to important details being overlooked (Gilson et al., 2020; Alami et al., 2020).

Even automated or semi-automated approaches face limitations. Molla et al. (2024) observed that ambiguity and inconsistency in user stories can hinder accurate identification of actors, use cases, and relationships, often requiring manual refinement for complex scenarios. Jahan et al. (2024) noted that rule-based methods are rigid and fail when requirements deviate from expected patterns, while generative AI methods can misinterpret actor interactions, raising concerns over semantic correctness. Similarly, Muhammad Ramzan et al. (2024) highlighted that class diagram generation is affected by ambiguous or domain-specific requirements, necessitating post-processing or human validation. Ferrari et

al. (2024) added that large language models, while producing syntactically correct diagrams, may generate semantically incorrect or hallucinated elements, emphasising the importance of careful prompt design and validation.

Certain challenges are specific to particular model types. For example, goal models are highly labour-intensive, as they require careful analysis to capture objectives, sub-goals, and dependencies accurately (Gunes et al., 2021). In UML, goal models are typically represented using use case diagrams, which illustrate actors, their goals, and the relationships between those goals. Large volumes of user input further complicate the transformation process, especially when generating UML diagrams that require structured relationships between actors, use cases, and associations. This combination of complexity and data volume highlights the need for automation or semi-automation supported by NLP techniques to ensure both accuracy and efficiency.

Furthermore, there are recent studies on the use of NLP-based tools and hybrid frameworks, such as UMLify and NL2Code that can automate parts of the UML generation process but still struggle with accurately capturing nuanced relationships or domain-specific elements (Tejaswini et al., 2025; Nair & Thushara, 2025). Likewise, Siddeshwar (2025) demonstrated that even large language models require extensive training and dataset annotation to correctly infer goals, sub-goals, and actor interactions from user stories. Babaalla et al. (2025) also noted that the lack of standardized annotated datasets limits the performance of deep learning approaches for UML element extraction, making post-processing and human validation essential.

Table 2.1 summarises the most frequently reported challenges across the reviewed studies. Manual diagram generation was consistently noted as demanding significant labour

and domain expertise, making the process both costly and time-consuming (Abdelnabi et al., 2021; Alami et al., 2020; Fischbach et al., 2020; Gunes et al., 2021; Maatuk & Abdelnabi, 2021). This difficulty is compounded by the complexity and ambiguity inherent in natural language, as textual user stories often contain redundant, incomplete, or inconsistent information, which increases the likelihood of errors when extracting actors, use cases, and their interactions (Javed & Lin, 2018, 2021; Abdelnabi et al., 2021; Vasques et al., 2019; Ternes et al., 2021). Even automated or AI-assisted techniques, while reducing manual effort, are not without limitations; they can misidentify relationships or produce semantically incorrect elements, particularly when requirements are complex or deviate from standard patterns (Jahan et al., 2024; Molla et al., 2024; Muhammad Ramzan et al., 2024; Ferrari et al., 2024). Collectively, these findings emphasise that, despite advances in NLP and AI, both human oversight and careful methodological design remain essential for accurate and reliable UML and conceptual model generation.

Overall, the findings demonstrate that, despite advances in NLP and AI-assisted modelling, manual effort, natural language complexity, and semantic challenges remain critical obstacles. These challenges collectively underscore the necessity of automation or semi-automation to support accurate, efficient, and scalable UML and conceptual model generation in Agile environments.

Table 2.1: Challenges associated with designing a UML model or another conceptual model

No	Challenges/Problems	Primary study
1	High time consumption	(Abdelnabi et al., 2021; Alami et al., 2020; Fischbach et al., 2020; Maatuk & Abdelnabi, 2021)
2	Require lots of effort	(Abdelnabi et al., 2021; Alami et al., 2020; Fischbach et al., 2020; Gunes et al., 2021; Maatuk & Abdelnabi, 2021)
3	High cost	(Abdelnabi et al., 2021; Alami et al., 2020; Fischbach et al., 2020; Maatuk & Abdelnabi, 2021)
4	Complexity of natural language	(Abdelnabi et al., 2021; Javed & Lin, 2018; Javed & Lin, 2021)
5	Ambiguity of natural language	(Abdelnabi et al., 2021; Javed & Lin, 2018; Javed & Lin, 2021)
6	Information duplicity, incompleteness, redundancy, and ambiguity	(Vasques et al., 2019)
7	Actors and their actions confusion	(Vasques et al., 2019)
8	Difficulty to identify important identifiers	(Ternes et al., 2021)

Table 2.1 continued

9	Process of manually transforming textual user stories to UML model is error prone due to high volume of textual user stories	(Alami et al., 2020)
10	Missing important information from textual user stories	(Gilson et al., 2020)
11	Ambiguity in user stories, misidentification of relationships, manual refinement needed	(Molla et al., 2024)
12	Rule-based rigidity, AI misinterpretation, semantic correctness of diagrams	(Jahan et al., 2024)
13	Ambiguous requirements, inaccurate relationship extraction, manual validation	(Muhammad Ramzan et al., 2024)
14	Semantic errors in LLM-generated diagrams, hallucinated elements, need for prompt engineering	(Ferrari et al., 2024)
15	NLP-based tool limitations in capturing nuanced relationships	(Tejaswini et al., 2025)
16	NLP/hybrid frameworks require post-processing or human validation	(Nair & Thushara, 2025)

Table 2.1 continued

17	Large language models require extensive training and annotated datasets for accurate goal and actor extraction	(Siddeshwar, 2025)
18	Lack of standardised annotated datasets limits deep learning model performance for UML element extraction	(Babaalla et al., 2025)

2.3.2 NLP tools and techniques used to assist the generation of a UML model from textual user stories

Natural Language Processing (NLP) tools and techniques are crucial for converting unstructured textual requirements, such as user stories, into structured UML or conceptual models. By systematically extracting actors, use cases, relationships, classes, and attributes, these tools facilitate the accurate and efficient generation of diagrams.

Table 2.2 summarises the NLP tools used across the primary studies. Stanford CoreNLP was the most frequently employed tool, appearing in twelve studies. Its suite of features, including tokenisation, part-of-speech (POS) tagging, lemmatisation, dependency parsing, and Open Information Extraction (OpenIE), supports the generation of multiple diagram types. For example, POS tagging and tokenisation are essential for identifying actors and use cases in use case diagrams, while dependency parsing and OpenIE allow the correct mapping of interactions for sequence diagrams. Lemmatisation aids in standardising terminology, which is particularly valuable for class diagrams where attributes and class names must be consistent.

spaCy, used in four studies, was preferred for its lightweight and flexible framework. Its syntactic and dependency parsing capabilities were particularly effective in generating goal models and robustness diagrams, where understanding hierarchical relationships and actor responsibilities is critical. TreeTagger Parser and MADA+TOKAN were applied for specific linguistic requirements, such as morphological analysis and processing Arabic text, ensuring accurate extraction of actors and actions in non-English user stories. Other tools, including Word2Vec, WordNet, DeepSRL, Apache OpenNLP, and Stanford NL Parser, were employed to handle semantic relationships, phrase-level meaning, or specialised domain-specific language, which are necessary for more complex diagrams such as class diagrams or activity diagrams with conditional flows.

Recent studies in 2025 have expanded the landscape of NLP tools for UML generation. Hybrid frameworks such as UMLify (Tejaswini et al., 2025) and NL2Code (Nair & Thushara, 2025) combine traditional NLP preprocessing with rule-based and model-driven approaches to automate the generation of UML diagrams, including class, sequence, and use case diagrams. These tools incorporate tokenisation, dependency parsing, semantic analysis, and large language models to map textual requirements to structured diagrams. Siddeshwar (2025) demonstrated that even large language models require extensive training and dataset annotation to correctly infer goals, sub-goals, and actor interactions from user stories. Babaalla et al. (2025) further highlighted that the lack of standardized annotated datasets limits the performance of deep learning models for UML element extraction, making post-processing and human validation essential.

Table 2.3 outlines the NLP techniques applied in these studies. Tokenisation and POS tagging were foundational techniques used across almost all studies to identify candidate nouns and verbs corresponding to actors, use cases, or classes. Lemmatisation and stemming

helped unify terminology, reducing redundancy when generating class or sequence diagrams. Dependency parsing and syntactic analysis were critical for accurately establishing relationships between actors, use cases, and actions, supporting the construction of sequence diagrams and goal models. Semantic role labelling (SRL), applied in studies such as Schlutter and Vogelsang (2020), was particularly valuable for mapping actions to responsible actors in both use case and sequence diagrams. Phrase extraction and machine learning classification, as used by Muhammad Ramzan et al. (2024), enabled the identification of classes, attributes, and associations for class diagrams, even in the presence of domain-specific terms or ambiguous requirements.

Synthesising the findings from Tables 2.2 and 2.3, a clear relationship between tools, techniques, and diagram types emerges. Use case diagrams benefit most from tokenisation, POS tagging, and dependency parsing, which together allow for accurate extraction of actors, use cases, and their interactions. Sequence diagrams require advanced parsing and semantic analysis to identify message flows and interaction sequences. Class diagrams rely heavily on lemmatisation, phrase extraction, and domain-specific NLP techniques to define classes, attributes, and associations consistently. Goal models and robustness diagrams, in contrast, demand syntactic and dependency parsing to reveal hierarchical or goal-oriented relationships between system elements and actors.

Overall, the evidence demonstrates that the effectiveness of UML generation depends not only on the choice of NLP tool but also on the depth and combination of techniques applied. Tools with richer capabilities, such as Stanford CoreNLP and SpaCy, when paired with layered techniques including dependency parsing, semantic analysis, and SRL, consistently produce more complete and semantically accurate diagrams. Conversely, tools with narrower capabilities or studies relying on basic NLP techniques alone face challenges

in addressing ambiguity, complex sentence structures, and domain-specific terminology. Integrating multiple tools and techniques is therefore essential for automating or semi-automating UML and conceptual model generation, improving accuracy, efficiency, and scalability across diagram types.

Table 2.2: NLP tools used in primary studies

No	NLP Tool	Primary study
1	TreeTagger Parser	(Elallaoui et al., 2018)
2	Stanford CoreNLP	(Abdelnabi et al., 2021; Alashqar, 2021; Allala et al., 2019; Athiththan et al., 2018; Maatuk & Abdelnabi, 2021; Javed & Lin, 2018; Javed & Lin, 2021; Lano et al., 2021; Nasiri et al., 2020; Nasiri et al., 2021; Schlutter & Vogelsang, 2020; Shweta et al., 2018)
3	MADA + TOKAN	(Alami et al., 2020)
4	spaCy	(Gilson & Irwin, 2018; Gilson et al., 2020; Gunes & Aydemir, 2020; Kochbati et al., 2021)
5	Word2vec	(Kochbati et al., 2021)
6	Apache OpenNLP	(Lano et al., 2021)
7	DeepSRL	(Schlutter & Vogelsang, 2020)
8	WordNet	(Nasiri et al., 2020)
9	Stanford NL Parser	(Tiwari et al., 2019)
10	Hybrid NLP framework (UMLify)	(Tejaswini et al., 2025)
11	Hybrid NLP & Model-Driven Framework (NL2Code)	(Nair & Thushara, 2025)

Table 2.2 continued

12	Large Language Models (LLMs)	(Siddeshwar, 2025; Ferrari et al., 2024)
13	Deep Learning with Annotated Datasets	(Babaalla et al., 2025)

Table 2.3: NLP techniques used in primary studies

No	Primary Studies	NLP Techniques
1	(Elallaoui et al., 2018)	POS Tagging
2	(Abdelnabi et al., 2021)	Tokenisation, POS Tagging, Lemmatisation and Stemming, Parse Tree and Type Dependencies, OpenIE
3	(Tiwari et al., 2019)	POS Tagging, Type Dependencies
4	(Alami et al., 2020)	Tokenisation, POS Tagging, Disambiguation, Discretisation, Morphological Disambiguation, And Stemming, Lemmatisation
5	(Maatuk & Abdelnabi, 2021)	Tokenisation, POS Tagging, Lemmatisation and Stemming, Parse Tree and Type Dependencies, OpenIE
6	(Gilson et al., 2020)	Stemming And Lemmatisation, POS Tagging, Dependencies Tree
7	(Javed & Lin, 2018; Javed & Lin, 2021)	Tokenisation, Lemmatisation, POS Tagging
8	(Alashqar, 2021)	Tokenisation, POS Tagging, Dependency Parsing
9	(Gunes & Aydemir, 2020; Allala et al., 2019)	Tokenisation, POS Tagging
10	(Kochbati et al., 2021)	Word level semantic, tokenisation

Table 2.3 continued

11	(Nasiri et al., 2021)	Text Splitting, Tokenisation, POS Tagging, Lemmatisation, Type Dependencies
12	(Lano et al., 2021)	POS Tagging, Syntax Tree
13	(Gilson & Irwin, 2018)	POS Tagging, Named-Entity Recognition
14	(Schlutter & Vogelsang, 2020)	Tokenisation, Sentence Splitting, POS Tagging, Lemmatisation, Dependency Parsing, Coreference Resolution, Semantic Role Labelling
15	(Shweta et al., 2018)	Sentence Splitting, Text Parser, Universal Dependency, Lemmatisation
16	(Nasiri et al., 2020)	Tokenisation, POS Tagging, Conference Resolution, Stemming
17	(Molla et al., 2024)	Tokenisation, POS Tagging, Dependency Parsing
18	(Jahan et al., 2024)	Tokenisation, POS Tagging, Syntactic & Semantic Analysis, Rule-based Heuristics
19	(Muhammad Ramzan et al., 2024)	Tokenisation, Lemmatisation, Phrase Extraction, Machine Learning-based Classification
20	(Ferrari et al., 2024)	Tokenisation, POS Tagging, Lemmatisation, Large Language Model Prompting
21	(Tejaswini et al., 2025)	Tokenisation, Dependency Parsing, Semantic Analysis, Rule-based Mapping
22	(Nair & Thushara, 2025)	Tokenisation, POS Tagging, Dependency Parsing, Model-Driven Transformation
23	(Siddeshwar, 2025)	Tokenisation, POS Tagging, Semantic Parsing, LLM Fine-tuning
24	(Babaalla et al., 2025)	Tokenisation, POS Tagging, Deep Learning-based Classification using Annotated Datasets

Table 2.4: Frequency of NLP tools used in primary studies

No	NLP Tool	Frequency
1	Stanford CoreNLP	12
2	spaCy	4
3	MADA+TOKAN	1
4	Word2Vec	1
5	Stanford NL Parser	1
6	Apache OpenNLP	1
7	DeepSRL	1
8	WordNet	1
9	TreeTagger Parser	1
10	Hybrid NLP frameworks (UMLify, NL2Code)	2
11	Large Language Models (LLMs)	2
12	Deep learning with annotated datasets	1

2.3.3 Measuring the accuracy UML models or conceptual models that are generated from textual user stories

The primary concern of the third RQ is to study the validation method used in primary studies to evaluate the model generation. Besides, the type of UML model or any other conceptual models that are usually generated from textual user stories are also studied.

Use case diagrams can be generated from a set of textual user stories. Elallaoui et al. (2018) used the TreeTagger parser to perform NLP on the textual user stories, where the cleaned textual user stories were pre-processed and tagged with POS tags provided by this tool. A set of algorithms are used to extract the elements of the use case diagram which are the actors, use cases, and the association relationship. The accuracy of the approach in this research is measured using a validation through the precision and recall scores. In order to perform this validation, the automatically extracted elements are compared to the manually extracted elements, resulting in the categorisation of true positive (TP), false positive (FP), and false negative (FN) for the actors, use cases, and association relationships. This research does not fully generate every element of the use case diagram, which obviously can be seen through the types of relationships. The only types of relationships that can be generated through the approach are the association relationship. However, the accuracy of the elements generated are satisfactory, where the value for precision and recall involving the actors are 98% each, 87% precision value for both use cases and relationships, and 85% recall value also for both use cases and relationships.

Abdelnabi et al. (2021) proposed an NLP-driven approach for generating behavioral UML models, including sequence and collaboration diagrams, from natural language requirements. Their method employs tokenisation, part-of-speech tagging, lemmatisation, dependency parsing, and OpenIE through Stanford CoreNLP, combined with heuristic rules to extract key diagram elements such as actors, senders, receivers, and messages. These elements are subsequently visualised using UML drawing tools. The approach was validated through a case study on a Qualification Verification System (QVS), where the extracted elements were compared against manually constructed reference diagrams. The evaluation focused on the correctness and completeness of the generated elements, assessing whether

all relevant actors and messages were accurately captured and represented. Additionally, the study highlighted the utility of the method in supporting the generation of other UML diagrams, including class diagrams, use case diagrams, and activity diagrams, demonstrating its broader applicability in automated UML model generation.

Using an NLP tool called MADA + TOKAN, sequence diagrams are generated from Arabic user requirements which was done in (Alami et al., 2020). Besides, the use of an NLP tool, these researchers also applied a set of heuristic rules to obtain the sequence diagrams components which are the participants, messages, and workflow transitions. The participants of the sequence diagram that are identified in this approach include the sender, the main actor, and the receiver. The accuracy of their method was evaluated through experiments involving a set of case studies, which were validated by both students and industry professionals familiar with sequence diagrams. The results showed that the proposed framework was able to identify participants more accurately than students and achieved results comparable to those of experts. This means that the approach is able to correctly and consistently identify the entities involved in the system, reducing errors such as missing actors, misclassification of roles, or confusion between senders and receivers. However, the approach was less effective in improving the identification of messages, as the performance was similar to that of students and did not surpass the accuracy of expert validation. This suggests that while the framework strengthens participant detection, further refinement is needed to reliably extract messages and workflow transitions from textual requirements.

Furthermore, a set of heuristic rules are used along with the Stanford CoreNLP for generating use case diagrams and activity diagrams (Maatuk & Abdelnabi, 2021). Stanford CoreNLP is used to pre-process the requirements, while the heuristics rules are used to extract the elements of use case diagrams and sequence diagrams. The rules used for

identifying the elements of use case diagrams are actors' identification rules, use cases identification rules, and relationship identification rules. While for the identification of activity diagram elements, the rules used are activity diagram identification rules, decision node identification rules, action name identification rules, and activity group identification rules. This research uses the same method of validation as done by Abdelnabi et al. (2021) which is case study and comparison study. For the results of the experiment using the case study, their approach is able to generate class diagram, use case diagram, and activity diagram.

Validation approaches have also involved combining traditional NLP metrics with human-in-the-loop evaluation. Tejaswini et al. (2025) and Nair and Thushara (2025) evaluated hybrid frameworks such as UMLify and NL2Code using comparison studies against manually created diagrams, emphasising post-processing and expert validation to ensure correctness. Siddeshwar (2025) demonstrated that LLM-based goal and UML model generation requires extensive training and annotated datasets to achieve acceptable accuracy, while Babaalla et al. (2025) showed that the availability of standardised annotated datasets significantly improves deep learning performance for UML element extraction. These findings indicate a shift toward multi-metric and hybrid validation strategies to better assess both syntactic accuracy and semantic correctness in automatically generated UML models.

In addition, there is a proposed approach where the author plans to automatically generate sequence diagrams and class diagrams from scenario-based user requirements (Alashqar, 2021). This approach combines the use of Stanford CoreNLP with several NLP techniques, together with a set of algorithms. As for the validation used to measure the accuracy of the approach, the author is using a method that compares a manually generated sequence diagram and an automatically generated one by their approach. The accuracy of

the proposed approach ranges from 77% to 90%, where the actor achieved an accuracy score of 80%, the caller object achieved a score of 90%, the receiver object received a score of 88%, the operation achieved a score of 90%, and the messages achieved a score of 77%. The results are affected by the POS tags generated in the case study, as some actors cannot be identified. Furthermore, some of the primary studies do not include a validation method, while others include only their validation plans. To summarise the findings relevant to this RQ, the details for the rest of selected primary studies are shown in Table 2.5. This includes the generated UML diagram, validation method, and score for each of the approach in the primary studies.

Table 2.5: UML model or conceptual model generated by the primary studies

Primary Study	Model generated	Validation method	Score/Results
(Elallaoui et al., 2018)	Use case diagram	Precision and recall	Precision: (Actor = 98%, Use cases = 87%, Relationship = 87%) Recall: (Actor = 98%, Use cases = 85%, Relationship = 85%)
(Abdelnabi et al., 2021)	Sequence diagram and collaboration diagram	Case study and comparison study	Able to generate sequence diagram, collaboration diagram, class diagrams, use case diagrams, and activity diagrams

Table 2.5 continued

(Tiwari et al., 2019)	Use case scenarios	Case study and comparison study	86.1% completion 89.7% accurate 92.8% consistent 87% non-redundant
(Alami et al., 2020)	Sequence diagram	Case study and comparison study	Generate participants element better than students, but lesser in terms of messages elements
(Maatuk & Abdelnabi, 2021)	Use case diagram and activity diagram	Case study and comparison study	Able to generate class diagram, use case diagram, and activity diagram
(Gilson et al., 2020)	Robustness diagram	Case study and comparison study	Generate 81.4% syntactically valid diagrams
(Javed & Lin, 2018)	ERD	Precision, recall, and over generation	Recall = 95% Precision = 93 Over generation = 9% Overall accuracy = 96%
(Alashqar, 2021)	Sequence diagram, Class diagram	Case study and comparison study	Overall accuracy: 77% - 90%

Table 2.5 continued

(Gunes & Aydemir, 2020)	Goal model	Prototype demonstration	Planning stage
(Kochbati et al., 2021)	Use case diagram	Case study, precision, recall, and F-measure	Precision: (Actors = 100%, Use cases= 95% - 97%, Relationship = 75% - 97%) Recall: (Actors = 100%, Use cases = 89% - 94%, Relationships = 84% - 90%) F-measure: (Actors = 100%, Use cases = 93% - 94%, Relationships = 79% - 93%)
(Gilson & Irwin, 2018)	Robustness diagram	-	Planning stage
(Shweta et al., 2018)	Class diagram	Case study, precision, and recall	Precision = 82% Recall = 94%
(Nasiri et al., 2020)	Class diagram	Case study, comparison study, and precision	Overall precision = 98%
(Tejaswini et al., 2025)	Use case diagram, Class diagram	Case study and comparison study	Automated generation is effective, however, post-processing is required.

Table 2.5 continued

(Nair & Thushara, 2025)	UML diagrams and code	Case study and expert validation	Accurate element extraction but human validation needed
(Siddeshwar, 2025)	Goal models and UML diagrams	Dataset-based evaluation, expert validation	Accuracy improves with fine-tuned LLMs and annotated datasets
(Babaalla et al., 2025)	Class diagram	Precision, recall using annotated datasets	Performance limited by dataset quality but annotation improves accuracy

2.4 Review Discussion

This subsection discusses the key findings from the selected primary studies, based on the three research questions: challenges in model generation, NLP tools and techniques used, and the types and accuracy of the generated models.

2.4.1 Challenges in Model Generation (RQ1)

The review highlights that natural language complexity is the primary barrier to accurate model generation. Ambiguities, inconsistencies, and incomplete information in user stories create difficulties for both human analysts and automated approaches. The high reliance on manual interpretation in traditional methods not only increases the time and effort required but also introduces a greater likelihood of errors, particularly when processing large volumes of user stories. Studies utilising rule-based or AI-assisted methods demonstrate that while automation can mitigate some of these issues, semantic errors and misinterpretation of relationships remain critical concerns. This suggests that fully automated transformation

from user stories to UML models is still challenging, and human oversight remains essential. Furthermore, the type of UML or conceptual model influences the complexity of generation. For instance, use case diagrams, while relatively straightforward, require careful identification of actors and relationships. More behaviourally complex diagrams, such as sequence or collaboration diagrams, demand the extraction of interactions, messages, and temporal ordering, which is highly sensitive to linguistic nuances. The discussion reveals that Agile contexts amplify these challenges due to rapid iteration and evolving requirements, emphasising the need for approaches that can quickly adapt to changing user stories without sacrificing accuracy.

2.4.2 NLP Tools and Techniques (RQ2)

The widespread adoption of tools such as Stanford CoreNLP and spaCy reflects their versatility and capacity to support a full NLP pipeline, from tokenisation and part-of-speech tagging to dependency parsing and semantic role labelling. The discussion indicates that tool selection is largely guided by the language of the requirements and the target UML model. For example, Arabic requirements necessitate tools like MADA+TOKAN, while sequence diagram generation benefits from advanced syntactic and semantic analysis. A notable trend is the combination of traditional NLP techniques with heuristic rules or machine learning models. This hybrid approach balances structural extraction with domain-specific knowledge, addressing issues that purely statistical or generative models might miss. However, the review also highlights limitations in this integration, such as the rigidity of rule-based methods and the potential for large language models to hallucinate elements or produce semantically inaccurate diagrams. These insights underscore that, while NLP significantly enhances efficiency, careful calibration of tools and techniques is essential to achieve reliable model generation.

2.4.3 Types and Accuracy of Generated Models (RQ3)

Use case, class, and sequence diagrams remain the most frequently generated UML models from textual user stories, reflecting their centrality in requirements analysis and design. Other models, such as robustness diagrams, goal models, and ERDs, were less common, often appearing in studies exploring specialised domains or advanced analysis techniques. The discussion reveals that accuracy is highly dependent on both the NLP approach and the complexity of the diagram. Actor identification and simple structural relationships generally achieve high precision and recall, whereas behavioural interactions and semantic relationships are more prone to errors. Validation through case studies, expert comparison, or precision/recall metrics provides confidence in the feasibility of automated or semi-automated generation, yet also emphasises the limitations of current approaches. This suggests a need for iterative refinement, integration of domain knowledge, and potentially hybrid human-AI workflows to improve the robustness and usability of generated models.

2.4.4 The Objectives and Rationale of UML Use Case Diagrams in Agile Requirements Engineering

Within Agile development, textual user stories capture functional requirements from an end-user perspective, yet they often lack the formal structure necessary for systematic design and validation (Cohn, 2015). UML use case diagrams provide a standardised visual representation that bridges this gap by modelling actors, use cases, and their interactions within a defined system boundary (Seidl et al., 2015). The primary objective of integrating diagram generation within this research is to transform narrative requirements into an interpretable and reusable visual format that enhances clarity, supports stakeholder alignment, and maintains traceability back to original user stories (Elallaoui et al., 2018).

The rationale for this focus stems from several documented challenges: manual diagram creation is time-consuming and error-prone (Maatuk & Abdelnabi, 2021); informal user stories often lead to ambiguities that visual models can help resolve (Gilson et al., 2020); and maintaining consistency between evolving requirements and design artifacts is difficult without automated traceability (Javed & Lin, 2018). By automating the generation of UML use case diagrams from structured textual user stories, this research aims to address these issues, providing a consistent, standardised, and traceable visual model that supports both communication and validation throughout the software development lifecycle.

2.4.5 Implications and Future Directions

Overall, the discussion points to several implications for practice and research. In Agile environments, semi-automated generation of UML models can accelerate requirement validation and improve communication among stakeholders. Nevertheless, challenges such as semantic accuracy, domain-specific terminology, and evolving requirements necessitate continued research into adaptive, context-aware NLP methods. Future work may explore more sophisticated hybrid approaches, leveraging both AI-driven generative techniques and domain-specific heuristics, to produce reliable, comprehensive UML and conceptual models from natural language requirements.

2.5 Addressing Key Gaps in Prior Research

Prior research in requirements engineering and model-driven development has highlighted several challenges when attempting to automate or semi-automate the generation of UML diagrams from textual requirements:

i. Ambiguity in Natural Language: Textual user stories often contain domain-specific terminology, informal expressions, and varied syntactic structures, making them difficult to parse reliably.

ii. Lack of Standardised Formats: Even within Agile methodologies, textual user stories can differ in structure, potentially causing inconsistencies during automated parsing.

iii. Complex Relationships: While use cases typically depict functional goals, textual user stories sometimes imply more nuanced relationships such as include, extend, or generalisation, that require additional logical inference.

iv. Traceability and Validation: Ensuring that each element in the UML diagram accurately reflects stakeholder requirements remains a manual, error-prone process.

US2UCD addresses these gaps by integrating domain knowledge with robust NLP techniques and a rule-based inference engine. Specifically:

i. Advanced NLP: Tokenisation, part-of-speech tagging, lemmatisation, and dependency parsing provide the linguistic details needed to identify actors (nouns), actions (verbs), and objects.

ii. Rule-Based Reasoning: Customizable logical rules interpret the NLP outputs to distinguish between primary use cases, supporting use cases, actors, and relationships. Domain-specific rules can be added to capture specialised vocabulary and usage patterns.

iii. Model Generation: By automatically generating an XMI (XML Metadata Interchange) representation of the identified actors, use cases, and relationships, the framework seamlessly transitions to UML modelling tools for diagram rendering.

2.6 Summary

This chapter reviewed the background and related literature on the automatic generation of UML and other conceptual models from textual user stories using Natural Language Processing. It began by explaining Agile development, the role of textual user stories in capturing user requirements, and the importance of UML diagrams, particularly use case, sequence, and class diagrams, for structured representation and stakeholder communication. A systematic literature review was conducted using four major academic databases, resulting in 20 primary studies published between 2018 and 2025.

The review highlighted consistent challenges in model generation. Manual transformation of user stories into UML diagrams requires significant time and effort and is prone to errors due to the ambiguity, redundancy, and incompleteness of natural language. Even semi-automated approaches face limitations, such as incorrect identification of actors, relationships, and messages, and difficulties in handling domain-specific terminology or complex interactions.

NLP tools and techniques played a central role in addressing these challenges. Stanford CoreNLP and spaCy were the most frequently used tools, while MADA+TOKAN enabled processing of Arabic requirements. Techniques such as tokenisation, part-of-speech tagging, lemmatisation, dependency parsing, and semantic role labelling were applied, often in combination with heuristic rules or machine learning methods, to extract actors, use cases, messages, and relationships. The synthesis of findings from Tables 2.2 and 2.3 shows that the choice and combination of tools and techniques directly affect the accuracy and completeness of generated models.

Validation methods included precision and recall metrics, case study experiments, and expert comparison. Results indicate that actor and participant identification generally achieve high accuracy, whereas message extraction and complex interactions are more error prone. Some approaches like the framework by Alami et al. (2020), were able to identify participants more accurately than students and produced results comparable to experts, although message identification still required improvement.

Overall, the chapter demonstrates that NLP-based automation can improve efficiency and consistency in generating UML and conceptual models from textual user stories, but challenges remain in achieving semantic correctness and managing complex requirements. These findings highlight the need for approaches that combine linguistic analysis, heuristic rules, and domain knowledge to develop more robust, adaptive, and scalable methods for model generation in Agile software development.

CHAPTER 3

METHODOLOGY

3.1 Overview

This chapter is organised as follows. Section 3.1 describes the overview of the research methodology used for conducting this research. Section 3.2 explains the details of the research methodology and Section 3.3 concludes the research methodology.

3.2 Introduction to Research Methodology

This research is conducted based on four main stages which are Stage 1: Prior Investigation and Content Analysis, Stage 2: Identification of Model Design Considerations and Attributes, Stage 3: Further Investigation, and Stage 4: Evaluation of Framework. These stages are crucial parts of this research where it is used as the main guideline to identify process, procedures, and suitable techniques to be applied in order to achieve the research objectives. Details for each stage are portrayed in the following subsections.

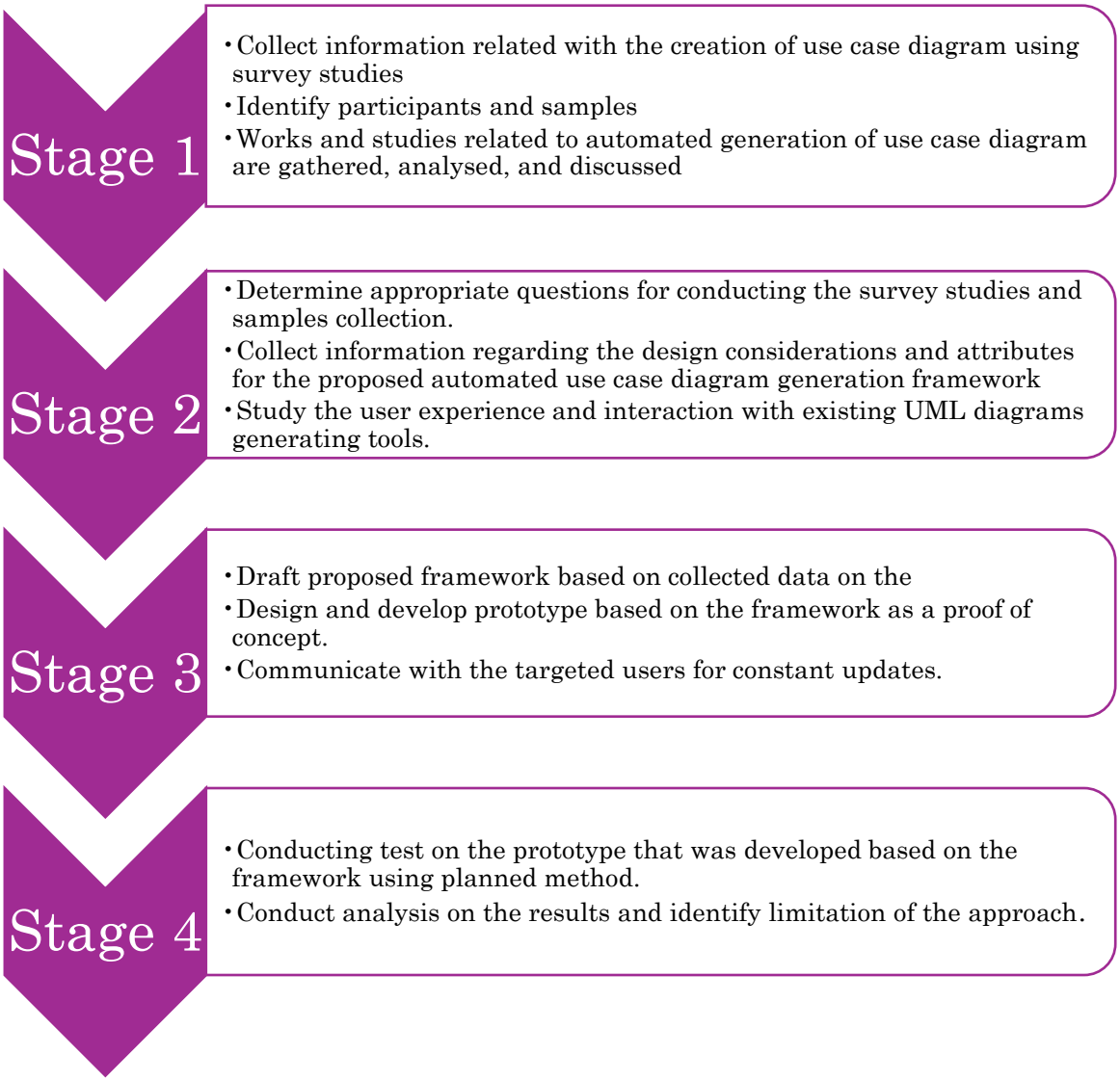


Figure 3.1: Four stages of the methodology used for conducting the research

3.2.1 Stage 1: Prior Investigation and Content Analysis

The primary objective of this stage was to develop a comprehensive understanding of the research topic, which is a critical foundation for subsequent stages of the study. Initially, data collection focused on the generation of conceptual models from textual user stories. This process was conducted through a combination of survey studies, direct observation of participants, and a systematic literature review.

The survey instrument was designed to elicit detailed information regarding the participants' practices in utilising conceptual models, particularly UML diagrams, during requirement documentation. The questionnaire targeted Software Engineering students at UNIMAS who were undergoing industrial training. It explored aspects such as the types of models employed, the perceived significance of these models in accurately representing requirements, and common challenges or limitations encountered during their application. Survey responses were systematically analysed to identify trends, recurring practices, and gaps in knowledge or methodology.

In parallel, observational studies were conducted to supplement the survey findings with qualitative insights. Selected participants were observed while performing tasks that involved the creation or reference of UML diagrams. Observation focused on the sequence of modelling steps, the selection and usage of UML elements, and the strategies employed to translate textual user stories into formal representations. Detailed field notes captured recurring challenges, inconsistencies, and adaptive strategies used by participants. This approach provided a nuanced understanding of real-world practices, which could not be fully captured through survey data alone.

The combined findings from the survey and observational studies informed both the preliminary investigations and the subsequent design of the US2UCD framework. These insights provided a baseline for evaluating the effectiveness and usability of the framework in later stages of the research. Concurrently, a systematic literature review was conducted to contextualise the study within existing research and to identify gaps in methodologies for transforming textual user stories into UML models. Figure 3.2 presents the procedural flow of this stage in pseudocode format.

```
Begin
    Define research problem
    Review related literature
    Design preliminary survey
    Distribute survey to participants
    Collect and analyse responses
    Identify gaps and limitations
End
```

Figure 3.2: Process flow of activities conducted in Stage 1

3.2.2 Stage 2: Identification of Model Design Considerations and Attributes

This stage aimed to refine the initial data collection instruments and gain deeper insights into user needs, experiences with existing tools, and the manual creation of conceptual models such as use case diagrams. While the research focuses on use case diagrams, investigating other UML models such as sequence diagrams, activity diagrams, and class diagrams, provides a broader context on modelling practices, common challenges, and workflow patterns. Understanding how participants interacted with these related models helped ensure that the proposed framework would be robust, compatible with typical modelling approaches, and capable of addressing recurring issues in the software design process.

Information gathered from Stage 1 revealed limitations in the preliminary survey, which, while valuable for identifying general trends, lacked sufficient depth in capturing participants' detailed workflows, challenges, and preferences. Some questions in Stage 1 were ambiguous or too broad, leading to incomplete or inconsistent responses. Additionally, Stage 1 primarily focused on identifying whether participants used conceptual models, rather than exploring how and why they interacted with these models in practice.

To address these gaps, the survey instrument was revised and expanded for Stage 2. The refined questionnaire was more comprehensive and structured, explicitly designed to

capture nuanced information relevant to the proposed framework. It included scenario-based questions that required participants to demonstrate how they would model textual user stories into UML diagrams, thereby providing insight into actual modelling practices. Likert-scale items measured user satisfaction and perceived effectiveness of current tools and approaches. Open-ended questions allowed participants to articulate challenges, suggest improvements, and provide qualitative context to their quantitative responses. Compared to Stage 1, the Stage 2 survey offered greater specificity, coverage, and clarity, ensuring that the collected data would support more informed design decisions for the framework.

The second activity of this stage involved evaluating specific design considerations and attributes for the framework. Participants were asked to rate the importance of clarity, traceability, scalability, and usability. Clarity assessed the ease with which models could be interpreted by both technical and non-technical stakeholders. Traceability ensured that each model element could be linked back to its corresponding user story. Scalability measured the framework's ability to handle systems of varying complexity, and usability evaluated the intuitiveness and simplicity of the tool. Collecting these ratings helped prioritise features in the framework's design and ensured alignment with user expectations.

The third activity looked at how participants used existing UML tools, specifically Visual Paradigm and Enterprise Architect. Participants were asked to complete tasks such as creating use case diagrams from textual user stories and updating existing diagrams. While they worked, the researcher observed their actions and took notes on how they used the tools, the steps they followed, and any problems they faced. Key points recorded included the time taken to finish tasks, mistakes made, and how participants tried to keep their diagrams consistent. Participants were also asked to think aloud as they worked, explaining what they were doing and why. This helped to understand their thought process and the difficulties they

encountered. Screen recordings and notes were taken for later review to ensure no detail was missed. After the tasks, participants completed a short questionnaire to give their feedback on the tools, including what they found easy or difficult and suggestions for improvement. This method of observation and feedback helped to identify common challenges such as steep learning curves, limited automation, and difficulties in keeping diagrams consistent. The findings were used to guide the design of the US2UCD framework, focusing on automation, clarity, and features that support users in creating diagrams more easily.

In summary, Stage 2 extended the findings of Stage 1 by addressing limitations identified in the preliminary survey, increasing the depth and breadth of the questions, incorporating both quantitative and qualitative measures, and examining related UML models to provide a comprehensive context. The insights obtained in this stage directly informed the drafting of the framework in Stage 3, ensuring that it would be practical, user-friendly, and responsive to real-world modelling challenges. Figure 3.3 illustrates the overall flow of activities conducted in this stage.

```
Begin
    Refine data collection instruments
    Collect user experience feedback
    Identify modelling challenges
    Extract design attributes
    Prioritise attributes based on importance
    Define framework design requirements
End
```

Figure 3.3: Process flow of activities conducted in Stage 2

3.2.3 Stage 3: Further Investigation

The third stage transitions from data collection to framework development and prototype creation. This stage aims to transform the findings from Stage 2 into a tangible, functional framework that can generate conceptual models from textual user stories. By

using the insights gathered from surveys and observations, the framework was designed to address real user challenges, such as reducing repetitive tasks, improving clarity, and maintaining consistency across diagrams. The prototype incorporated features prioritised by participants, including automation of routine steps, clear labelling of elements, and traceability of model components back to their corresponding user stories. These design choices ensured that the framework would be practical to use in real workflows, intuitive for both technical and non-technical users, and closely aligned with the needs and expectations observed in Stage 2.

The first activity in this stage was drafting the framework. The framework was designed as a structured, three-step process consisting of an Input Module, a Processing Engine, and an Output Module. The Input Module accepts textual user stories in a standardised format. This standardised format is defined as a template that clearly separates the actor, the action or goal, the expected outcome, and any conditions or dependencies. The Processing Engine uses rule-based algorithms to parse the textual user stories and map them to model components such as actors, use cases, and relationships. The Output Module generates initial UML diagrams, which can then be edited or refined by users. Logical mapping rules were documented to ensure that the process remained consistent, accurate, and traceable to input textual user stories.

To formally represent the logical flow of the proposed framework, a high-level pseudocode was developed which is shown in Figure 3.4. This pseudocode describes the sequence of operations performed when transforming textual user stories into UML use case elements. It provides an abstract and implementation-independent view of the framework's core logic, ensuring transparency, reproducibility, and consistency in model generation. The pseudocode served as a reference model during prototype development and ensured that the implemented system followed the designed framework logic.

```
Input: Set of structured textual user stories
Output: UML use case diagram

Begin
    Read textual user stories
    Pre-process text (tokenisation, lemmatisation, POS tagging,
dependency parsing)

    For each user story do
        Identify actor from subject
        Identify use case from main verb phrase
        Identify relationships (include, extend, association)
        Store extracted elements
    End for

    Apply logical rules to validate elements
    Resolve duplicates and inconsistencies
    Generate UML use case diagram
    Link each diagram element to its source user story

End
```

Figure 3.4: Logical flow of proposed framework

The second activity involved the development of a prototype to demonstrate the feasibility of the framework. The prototype was implemented using Python for NLP and diagram generation. Early iterations of the prototype focused on automating basic tasks, such as identifying actors and use cases based on keywords in textual user stories. Visualisation libraries were used to render UML diagrams. The prototype served as a proof of concept,

showcasing the potential of the framework to simplify and accelerate the model generation process.

The third activity involved iterative refinement based on user feedback. Selected participants from Stage 2 were invited to interact with the prototype and provide feedback on its usability, clarity, and accuracy. Regular feedback loops ensured that issues such as unclear outputs or missing relationships were addressed promptly. For example, participants suggested improvements to the UI, such as better diagram customisation options and clearer traceability between textual user stories and model elements.

By the end of this stage, a functional prototype was developed, demonstrating the practical application of the framework. This prototype was ready for evaluation in Stage 4. In addition, the logical flows of involved process in this stage are shown in Figure 3.5.

```
Begin
    Design framework modules
    Define NLP and mapping rules
    Implement processing engine
    Generate UML diagrams
    Collect user feedback
    Refine framework logic
End
```

Figure 3.5: Process flow of activities conducted in Stage 3

3.2.4 Stage 4: Evaluation of Framework

The final stage focuses on evaluating the framework's effectiveness through structured testing, result analysis, and identification of limitations, ensuring that the framework meets its objectives and provides actionable insights for improvement. Structured testing refers to a formal approach in which predefined procedures, criteria, and input scenarios are applied to ensure consistency and reliability in the evaluation results. In this study, structured testing was implemented through controlled experiments, designed to

systematically assess the performance, accuracy, and usability of the framework under controlled conditions. During these experiments, participants, consisting of software engineering students and academic practitioners with prior experience in UML modelling, were asked to input textual user stories of varying complexity into the prototype and examine the generated UML diagrams. The materials included the US2UCD framework, a set of textual user stories, and evaluation sheets to record both quantitative metrics such as precision, recall, F1-score, completeness, and task completion time, as well as qualitative observations related to usability, clarity, and traceability. The experimental protocol required participants to follow a standardised sequence of tasks to maintain consistency, allowing the researchers to isolate the framework's performance and identify strengths and limitations in a systematic and reproducible manner. Following data collection, the results were analysed using descriptive statistics to measure the framework's accuracy, completeness, and efficiency. The analysis revealed that the framework performed effectively for straightforward textual user stories but faced challenges when processing complex scenarios with multiple interdependencies. Additionally, participants noted limitations in the customisation options for generated diagrams. Overall, this stage validated the practical utility of the framework, highlighted areas for enhancement, and provided evidence to guide future refinements, ensuring that the system remains both effective and aligned with user needs. Figure 3.6 illustrates the process flow of activities conducted in this stage in the form of pseudocode.

```
Begin
    Design framework modules
    Define NLP and mapping rules
    Implement processing engine
    Generate UML diagrams
    Collect user feedback
    Refine framework logic
End
```

Figure 3.6: Process flow of activities conducted in Stage 4

3.3 Conclusion

This chapter has outlined the research methodology employed to achieve the objectives of this study, structured into four key stages which are Prior Investigation and Content Analysis, Identification of Model Design Considerations and Attributes, Further Investigation, and Evaluation of Framework. Each stage was systematically designed to ensure a robust, iterative, and user-centered approach to developing a framework for generating conceptual models from textual user stories.

CHAPTER 4

PRELIMINARY STUDY RESULT AND ANALYSIS

4.1 Overview

For the collection of preliminary data, a survey study has been conducted where the respondents can be grouped into two groups, which are industry professionals and internship students from Universiti Malaysia Sarawak (UNIMAS). The purpose of this survey study is to obtain ideas and information about the challenges faced by industry professionals such as Software Engineers, System Analysts, Software Developers, Requirement Engineers, as well as internship students from FCSIT, UNIMAS. This group of people is selected as participants in this survey study because they are regularly exposed to and deal with requirements design, which may involve UML Model generation. The survey was conducted using an online survey administration software, Google Forms, and the questions used were formulated according to the research questions. The questions used for conducting this survey is included in Appendix B.

Besides, a web-based application has been developed as part of the method to collect preliminary data. The web-based application will be used to collect UML use case diagram samples mainly from the software engineering students at UNIMAS. The samples collected will be used to conduct tests and evaluate prototypes which are done as proof of the framework's concept. This chapter will provide a detailed analysis that have been conducted to study the trends of UML diagram generation among Software Engineering students in UNIMAS.

4.2 Survey Results and Findings

This subsection will present the results gathered from the survey. Each result is described in detail, and its effect on this research was also elaborated.

4.2.1 Demographic Information

The demographic section describes the background of the respondents who participated in the preliminary study. A total of 155 software engineering students responded to the survey.

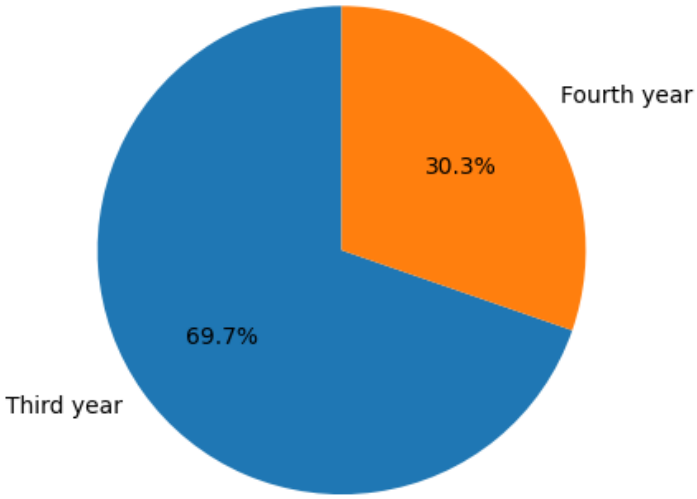


Figure 4.1: Distribution of respondents by year of study

As shown in Figure 4.1, the majority of respondents were third-year students (69.7%), followed by fourth-year students (30.3%). This indicates that most participants were actively involved in core software engineering courses and project-based learning, making them suitable respondents for evaluating issues related to UML modelling and automated tool support.

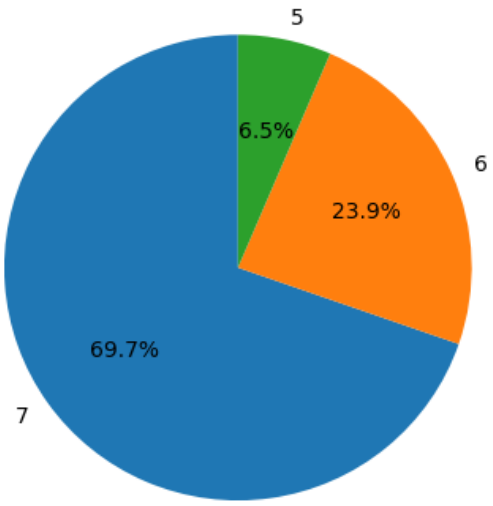


Figure 4.2: Distribution of respondents by group size

Based on pie chart in Figure 4.2, most respondents worked in groups of seven members (69.7%), which reflects the standard group configuration for the Software Engineering Laboratory course. Larger group sizes often increase coordination complexity, reinforcing the need for structured requirement documentation and UML-based modelling approaches.

4.2.2 Software Development Models in Use

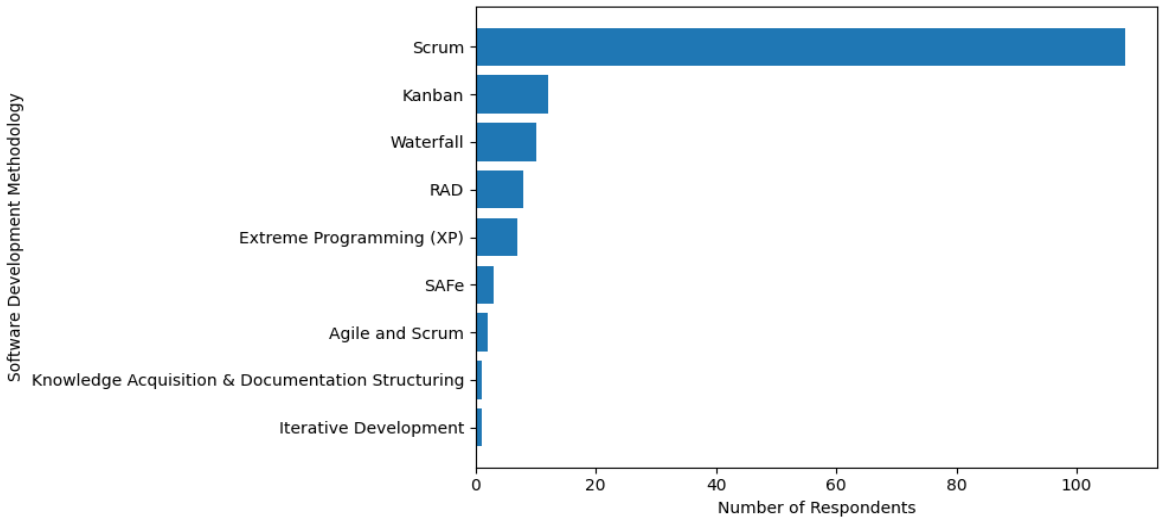


Figure 4.3: Results gathered for software development models used

Figure 4.3 presents a horizontal bar chart summarising the software development models currently employed by 153 survey respondents which are software engineering students in UNIMAS. Notably, Scrum emerges as the dominant methodology, with 70.6% of participants indicating that they use this framework in their projects. This substantial reliance on Scrum aligns with broader industry trends that favour Agile approaches due to their iterative nature, adaptability to changing requirements, and emphasis on continuous stakeholder feedback.

Despite the prominence of Scrum, other Agile methods also appear in the survey responses, albeit in smaller proportions. These include Extreme Programming (XP), Kanban, Lean Development, Crystal, and Feature Driven Development (FDD). The presence of multiple Agile frameworks highlights the diverse preferences and practices among teams, suggesting that different projects may require different types of Agile methodologies based on their unique goals, team dynamics, and technical requirements.

Interestingly, a segment of respondents still employs Waterfall which is a more traditional and plan-driven model. While it does not represent the majority, the continued use of Waterfall indicates that some teams may find value in its structured, sequential approach, particularly for projects with well-defined requirements, low tolerance for change, or contractual obligations that necessitate rigorous upfront planning.

Overall, these findings suggest that Agile methods, and Scrum in particular, have gained widespread acceptance as an effective means of managing modern software projects. Nonetheless, the variety of development models reported underscores the notion that no single methodology universally fits every project context. Instead, organisations and teams

often tailor their approach to suit specific project scopes, stakeholder needs, and resource constraints.

4.2.3 Requirements Gathering Methods

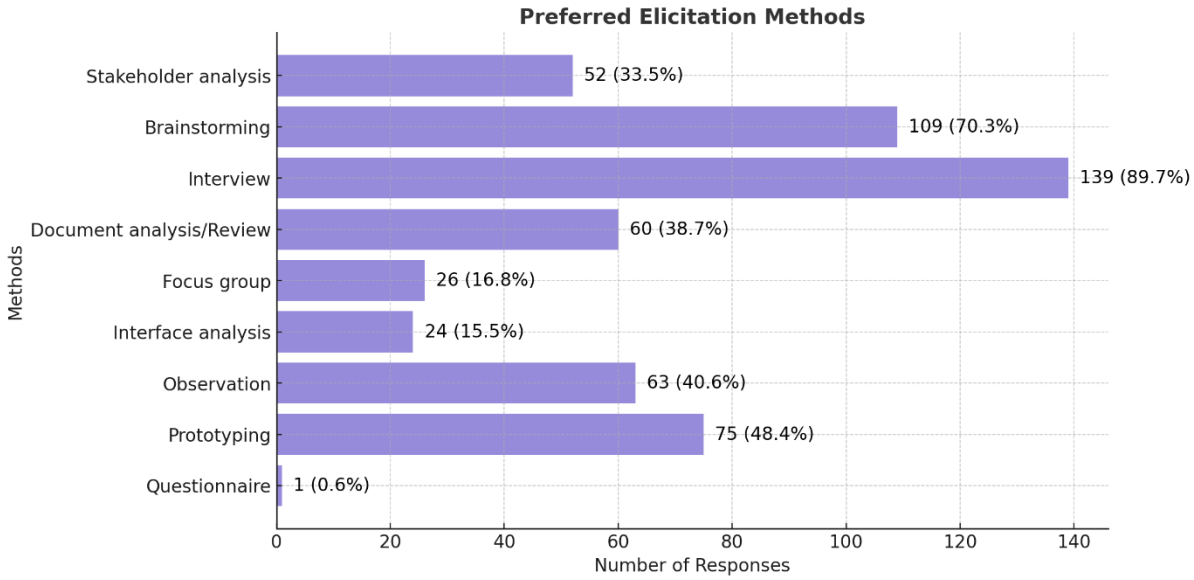


Figure 4.4: Results for requirements gathering methods

Figure 4.4 depicts the frequency of various requirement-gathering techniques utilised by 155 survey respondents. Notably, Interview emerges as the most prevalent approach, with 139 participants (89.7%) indicating its use. Brainstorming follows closely, employed by 109 participants (70.3%), while Prototyping ranks third at 75 responses (48.4%). Other frequently mentioned methods include Observation (63 responses, 40.6%) and Document Analysis/Review (60 responses, 38.7%), whereas techniques such as Focus Groups (26 responses, 16.8%) and Interface Analysis (24 responses, 15.5%) appear less commonly. Stakeholder Analysis accounts for 52 responses (33.5%), and Questionnaires register the lowest usage with 1 response (0.6%).

The prominence of Interviews underscores the value practitioners place on direct stakeholder engagement, allowing for rich, context-specific insights into user needs and

expectations. Such detailed qualitative data often translates well into textual user stories, which, in turn, can be systematically mapped to UML Use Case Diagrams. By capturing the interactions and objectives of various user roles (actors) through these stories, teams can visualise system functionalities more effectively.

Meanwhile, Brainstorming and Prototyping facilitate rapid ideation and iterative refinement of requirements. Through brainstorming, participants collaboratively explore a wide range of potential features, while prototyping offers a tangible representation of system elements, encouraging user feedback at an early stage. This iterative feedback loop can yield clearer, more comprehensive textual user stories, ultimately leading to more accurate UML use case representations.

Although less commonly employed, Observation and Document Analysis/Review remain valuable for verifying or complementing stakeholder narratives. By observing real-world user interactions or reviewing existing documentation, development teams can cross-check the accuracy of textual user stories and ensure alignment with established organisational processes and legacy systems. This multi-faceted approach helps in creating robust use cases that reflect actual operational environments.

In sum, the requirement-gathering methods identified in this study form the critical foundation for generating textual user stories, which then serve as inputs to UML use case diagrams. As such, understanding the relative emphasis on interviews, brainstorming, and prototyping as well as the supportive roles of observation, document review, and stakeholder analysis is vital for designing a comprehensive visualisation framework. This framework aims to bridge the gap between narrative user requirements and structured system models, ultimately enhancing clarity and communication in software development processes.

4.2.4 Challenges in Requirements Gathering

The following table encapsulates the key challenges encountered by students during the requirements elicitation phase. The survey conducted with student participants can be considered reliable within the context of this study. All participants were software engineering students with prior exposure to UML modelling and requirement analysis, which ensured that responses reflected informed experiences. This summary elucidates the nature of each challenge, its impact on the formulation of UML use case diagrams, and potential mitigation strategies.

Table 4.1: Challenges encountered by students for requirement elicitation

Challenge	Description	Impact on UML Use Case Diagrams	Proposed Mitigation
Incomplete or Vague Textual user stories	Textual user stories are often high-level and lack detailed descriptions.	Impedes precise definition of actors, system boundaries, and core use cases.	Employ structured elicitation techniques and use standardised templates.
Communication Barriers	Difficulties arise due to domain-specific jargon, language differences, and limited stakeholder access.	Leads to misinterpretations and incomplete capture of user requirements.	Enhance communication protocols through structured interviews and clarifying questions.
Stakeholder Uncertainty and Evolving Requirements	Stakeholders frequently exhibit uncertainty or change requirements during the project lifecycle.	Results in iterative revisions and inconsistencies within UML diagrams.	Implement iterative review sessions with continuous stakeholder engagement.

Table 4.1 continued

Limited Domain Knowledge	Students often possess inadequate understanding of the relevant business domain.	Produces incomplete or contextually inaccurate representations in UML models.	Provide domain-specific training and involve subject matter experts.
Time Constraints and Resource Limitations	Limited time and resources hinder comprehensive requirements gathering activities.	May lead to rushed elicitation processes and subsequently incomplete UML diagrams.	Allocate sufficient time and resources; utilise collaborative tools to support thorough analysis.
Inadequate Methodological Guidance	Absence of clear frameworks and guidelines for systematically eliciting requirements.	Contributes to inconsistent and poorly structured textual user stories.	Develop and adopt standardised methodologies for requirement elicitation.
Difficulty Capturing Non-Functional Requirements	Challenges exist in articulating qualitative aspects such as performance, security, and usability.	Critical system constraints may be underrepresented in the resulting UML models.	Incorporate focused sessions and checklists to systematically capture non-functional requirements.

As demonstrated in Table 4.1, each identified challenge not only disrupts the clarity of the elicited requirements but also adversely affects the precision and completeness of the resulting UML use case diagrams. Addressing these challenges is crucial for enhancing the overall fidelity of system modelling in the context of translating textual user stories into structured design representations. These challenges, derived from both quantitative and

qualitative data, have significant implications for the accuracy and efficacy of the subsequent modelling process using UML use case diagrams.

a) Incomplete or Vague Textual user stories

A prominent issue reported by participants was the tendency for textual user stories to be high-level and lacking in detail. This vagueness often complicated the translation of narrative requirements into precise UML elements, thereby affecting the definition of actors, system boundaries, and primary use cases.

b) Communication Barriers

Students frequently encountered obstacles in stakeholder communication, which impeded the clarity of the elicited requirements. The use of domain-specific terminology, language differences, and limited stakeholder availability contributed to misunderstandings and incomplete capture of user needs.

c) Stakeholder Uncertainty and Evolving Requirements

The dynamic nature of stakeholder expectations posed another significant challenge. Several respondents indicated that stakeholders were often unsure of their requirements, resulting in evolving textual user stories that necessitated continual revisions. This iterative process created difficulties in maintaining consistent and comprehensive UML use case diagrams.

d) Limited Domain Knowledge

A recurring challenge was the lack of sufficient domain expertise among the students. Without an in-depth understanding of the specific business processes and technical

constraints, students struggled to formulate detailed and contextually accurate textual user stories, thereby impacting the overall quality of the requirements.

e) Time Constraints and Resource Limitations

Tight project timelines and limited access to supportive tools or sufficient personnel were also cited as barriers. These constraints often resulted in rushed or incomplete stakeholder engagements, leading to the omission of critical details during the requirements gathering process.

f) Inadequate Methodological Guidance

Many participants expressed uncertainty regarding the systematic application of requirement elicitation techniques. The absence of structured guidelines or best practices hindered their ability to transform raw user input into well-defined textual user stories, ultimately affecting the precision of the resulting UML use case diagrams.

g) Difficulty Capturing Non-Functional Requirements

Finally, while functional requirements were generally well captured, there was a notable struggle in articulating non-functional aspects such as performance, security, and usability. The challenge of defining these qualitative parameters within the narrative framework of textual user stories further complicated their representation in UML models.

In summary, these findings underscore the complex interplay between communication, domain understanding, and methodological rigor in the requirements gathering process. Addressing these challenges is critical for enhancing the fidelity of textual user stories, thereby ensuring that the resulting UML use case diagrams accurately reflect stakeholder needs and system functionalities.

4.2.5 Requirement Documentation

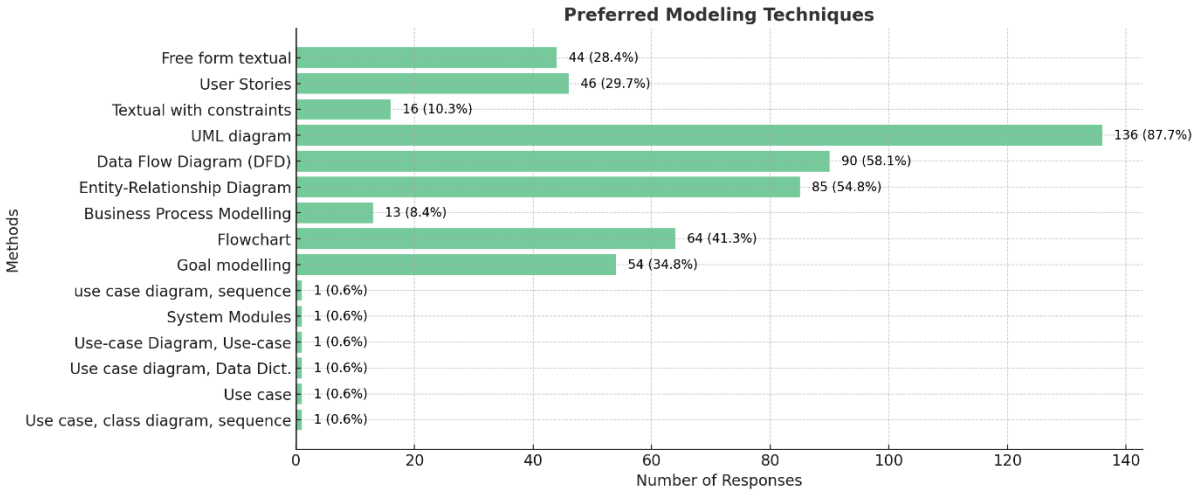


Figure 4.5: Requirements documentation techniques preferred

Figure 4.5 illustrates how the 155 survey respondents document requirements after the elicitation process. The data show that UML Diagrams are the predominant method, with 136 participants (87.7%) indicating their use. Data Flow Diagrams (DFDs) follow at 90 responses (58.1%), while ERD appear in 85 responses (54.8%). Other notable methods include Flowcharts (64 responses, 41.3%) and Business Process Modelling, where Goal Modelling and more specialised or composite approaches such as Use-case diagram, Sequence diagram, or System modules are used by a smaller fraction of respondents. Additionally, Textual user stories (46 responses, 29.7%) and Free-form Textual Documentation (44 responses, 28.4%) remain relevant, though less frequently employed than diagrammatic techniques.

4.2.5.1 Dominance of Diagrammatic Approaches

The high usage of UML Diagrams underscores their importance in conveying system structure and interactions in a standardised, visual manner. This preference suggests that students and practitioners alike value the clarity that UML can provide, particularly when

transitioning from narrative requirements like textual user stories to more formalised design representations.

4.2.5.2 Variation in Documentation Strategies

Despite the evident preference for UML, the survey reveals a diverse range of documentation practices, including Data Flow Diagrams (DFDs), ERD, flowcharts, and business process models. This diversity indicates that no single method universally addresses all project needs. Instead, teams often select techniques that best suit the complexity of the system, stakeholder preferences, and project constraints. Notably, Textual user stories continue to play a crucial role in capturing functional requirements from an end-user perspective, while free-form text allows flexibility in describing system aspects that may not fit neatly into standard diagramming conventions.

4.2.5.3 Relevance to Visualising Textual user stories

The presence of both Textual user stories and UML Diagrams among commonly used documentation methods highlights the central focus of this research: bridging the gap between narrative requirements and formalised, visual representations. By systematically translating textual user stories into UML use case diagrams, development teams can ensure that the high-level intentions of stakeholders are accurately reflected in the system's structural and behavioural models. This translation process helps maintain traceability and consistency throughout the software development lifecycle.

The findings from this survey underscore the necessity of a robust framework or methodology to guide the transformation of textual user stories into UML. Such a framework would address the challenges identified in previous sections such as vague requirements, communication barriers, evolving stakeholder needs as well as enabling students or practitioners to produce clearer, more coherent use case diagrams.

Ultimately, this research is significant because it promotes consistency and accuracy by providing structured guidelines for mapping narrative requirements to UML elements. This minimises misunderstandings and reduces design inconsistencies within development teams. Additionally, it enhances stakeholder communication by using visual models backed by well-defined textual user stories, facilitating clearer discussions about system functionalities and constraints.

Furthermore, it encourages best practices by integrating textual user stories with UML diagrams, supporting iterative refinement, traceability, and stakeholder validation in alignment with Agile principles and modern software engineering standards. Lastly, it addresses educational gaps by offering students a systematic approach to converting textual user stories into UML diagrams, bridging theoretical knowledge with practical application in software engineering. The educational gaps are addressed because students often struggle to translate high-level, narrative requirements into formal UML diagrams due to a lack of structured guidance or practical experience. By offering a systematic approach, the framework provides clear, step-by-step procedures for mapping textual user stories to UML use case elements such as actors, use cases, and relationships. This structured method helps students understand how to interpret stakeholder intentions, maintain consistency, and ensure completeness in their diagrams. Additionally, it bridges the gap between theoretical knowledge taught in lectures and practical application, giving students hands-on experience with a repeatable process that mirrors real-world software engineering practices. As a result, students gain both conceptual understanding and practical skills, improving their ability to create accurate, coherent, and traceable UML models from narrative requirements.

In summary, the survey results confirm the widespread reliance on diagrammatic methods for documenting requirements, while also highlighting the continued relevance of

textual user stories as a narrative means of capturing user needs. This intersection forms the crux of the present research, demonstrating both the demand for and the potential impact of a comprehensive method for visualising textual user stories in UML use case diagrams.

4.2.6 Problems Encountered During Requirement Documentation

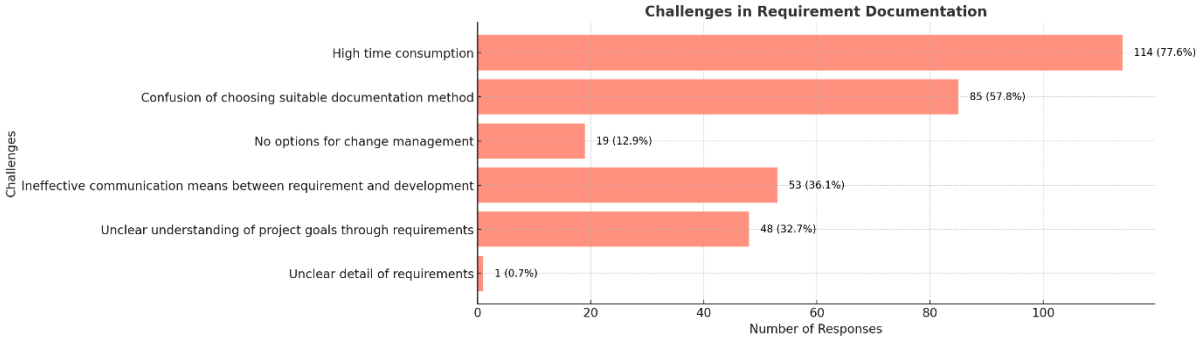


Figure 4.6: Common issues faced by respondents in requirement documentation

Figure 4.6 provides insights into the most common issues faced by 147 respondents when documenting their requirements. The data reveal several recurring challenges, each of which can significantly affect the accuracy, clarity, and maintainability of project documentation. These findings are particularly relevant in the context of transforming textual user stories into UML use case diagrams, as they underscore the need for efficient, structured, and adaptable documentation practices.

4.2.5.1 High Time Consumption

The most frequently cited challenge is the extensive amount of time required to document requirements. Participants noted that preparing thorough documentation, ensuring its accuracy, and revising it to reflect ongoing changes can be labour-intensive. In the context of translating textual user stories into UML, this time investment may increase further, as teams must carefully map narrative requirements to appropriate UML elements (actors, use cases, relationships).

4.2.5.2 Confusion in Selecting Suitable Documentation Methods

A significant proportion of respondents reported uncertainty in choosing the most appropriate documentation technique. Given the variety of tools and approaches ranging from textual specifications to UML diagrams, Data Flow Diagrams, and more teams often struggle to identify a method that balances completeness, ease of understanding, and adaptability to project changes. This confusion can impede the effective use of UML for visualising textual user stories, as suboptimal documentation methods may obscure the intended system behaviour.

4.2.5.3 Ineffective Communication Means

Nearly one-third of participants indicated difficulties in communicating between requirements and development teams. Inadequate or inconsistent communication channels lead to discrepancies between documented requirements and their actual implementation. Such gaps are especially critical when maintaining traceability between textual user stories and UML diagrams, as misunderstandings can result in misaligned models and system functionalities.

4.2.5.4 Unclear Understanding of Project Goals

Respondents also noted that incomplete or ambiguous project objectives hinder their ability to produce coherent documentation. When stakeholders and development teams lack a shared vision, textual user stories may become disjointed or contradictory, complicating the process of generating accurate UML models. Clarity of project goals is essential for ensuring that each use case correctly reflects end-user needs and overall system objectives.

4.2.5.5 No Options for Change Management

A smaller but notable group highlighted the absence of robust mechanisms to handle requirement modifications. In agile or iterative environments, the inability to track and manage changes systematically can lead to outdated documentation, misaligned UML diagrams, and potential rework. This shortcoming underscores the importance of version control and iterative review processes, which help maintain consistency between textual user stories and their corresponding models.

These challenges highlight the delicate balance between thoroughness, clarity, and flexibility in requirement documentation. For teams seeking to convert narrative textual user stories into formalised UML use case diagrams, the issues of time consumption, confusion in selecting methods, and communication barriers can be especially pronounced. This research thus aims to address these pain points by proposing a structured yet adaptable framework for documentation and visualisation. By doing so, practitioners can reduce ambiguity, enhance stakeholder alignment, and streamline the overall development lifecycle.

4.2.6 Creating Conceptual Models for Requirement Modelling

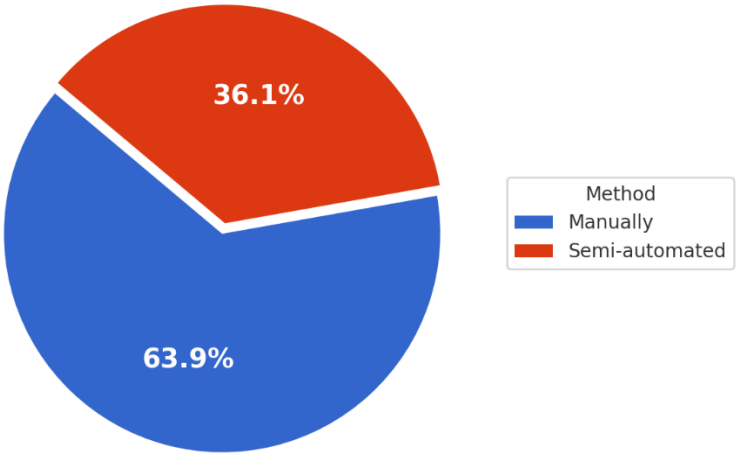


Figure 4.7: Approach used for diagram generation

Figure 4.7 illustrates the methods by which 155 respondents typically develop conceptual diagrams for requirement modelling. The majority, 63.9%, report creating these diagrams manually, while 36.1% utilise a semi-automated approach.

4.2.6.1 Prevalence of Manual Techniques

The predominance of manual diagram creation suggests that many students rely on conventional drawing tools or basic software applications such as standard UML diagramming tools without leveraging advanced automation features. This approach may stem from a desire for direct control over diagram elements or a lack of familiarity with more sophisticated tools. However, manual processes can be time-intensive and prone to human error, particularly when requirements change frequently.

4.2.6.2 Emerging Interest in Semi-Automation

Despite the continued reliance on manual methods, over one-third of respondents (36.1%) have adopted semi-automated techniques. These may include software tools capable

of partially generating diagrams from textual inputs like textual user stories or providing intelligent suggestions for model elements. By streamlining repetitive tasks, such tools can enhance efficiency and improve the consistency of documentation across different artifacts.

4.2.6.3 Relevance to Visualising Textual user stories

The mixed adoption of manual and semi-automated approaches highlights a key opportunity to refine the process of translating textual user stories into UML models. While manual methods offer flexibility and customisation, they can be labour-intensive, whereas semi-automated solutions can expedite diagram creation but often require specialised training or tool support. This research aims to bridge the gap by proposing a balanced methodology that reduces the overhead of manual modelling while ensuring that the unique nuances of textual user stories are accurately captured.

By integrating the strengths of both manual and semi-automated approaches, students and practitioners can address common documentation challenges such as time constraints and inconsistency, ultimately facilitating a more efficient and reliable workflow for requirement modelling.

4.2.7 Types of Conceptual Diagrams Created for Requirement Design

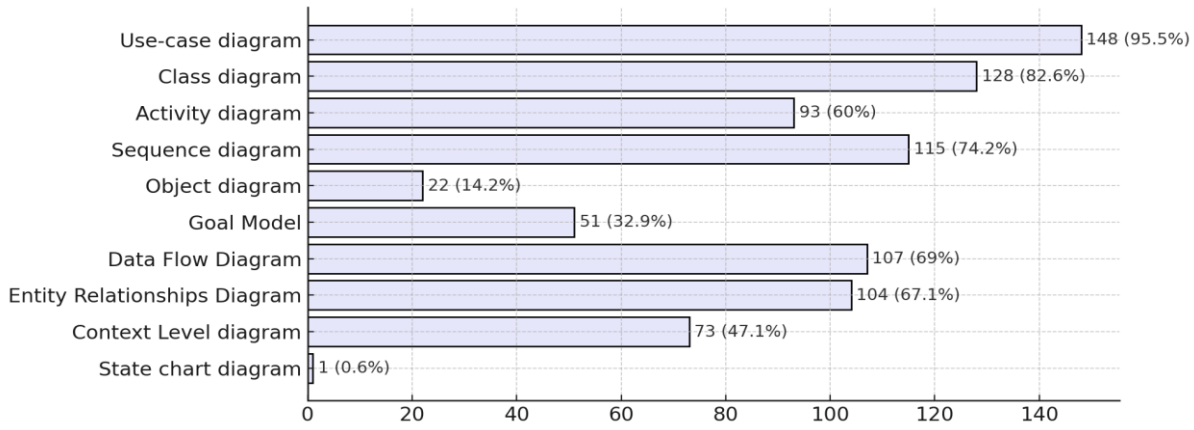


Figure 4.8: Types of conceptual diagrams used during requirement design

Figure 4.8 illustrates the types of conceptual diagrams that 155 respondents typically employ during the requirement design process. Notably, Use-Case Diagrams lead with 148 participants (95.5%) indicating their use, followed by Class Diagrams at 128 responses (82.6%), Sequence Diagrams at 115 responses (74.2%), and Activity Diagrams at 93 responses (60%). A range of other diagrams, including Data Flow Diagrams (107 responses, 69%), ERD (104 responses, 67.1%), and Context-Level Diagrams (73 responses, 47.1%), also feature prominently. By contrast, Object Diagrams (22 responses, 14.2%) and Goal Models (51 responses, 32.9%) are used less frequently, while State Chart Diagrams appear to be the least adopted, with only 1 response (0.6%).

4.2.7.1 Emphasis on UML and Related Models

The strong preference for Use-Case Diagrams reflects their central role in capturing functional requirements and user interactions within a system. This aligns with the broader trend of adopting UML-based methods, as evidenced by high usage rates for class, sequence, and activity diagrams, which offer standardised, widely recognised notations. Such notations

facilitate more transparent communication among stakeholders and more structured design processes.

4.2.7.2 Complementary Use of Other Diagramming Techniques

Although UML diagrams dominate, a significant portion of respondents also employ Data Flow Diagrams (DFDs), ERD, and Context-Level Diagrams. These methods, often associated with structured analysis or database design, indicate that teams tailor their modelling approach based on specific project needs. For instance, DFDs are well-suited for understanding data movement, while ERDs help define database schemas.

4.2.7.3 Underutilisation of Goal Models

The comparatively lower adoption of Goal Models suggests that many practitioners focus primarily on functional representations, potentially overlooking the advantages of modelling higher-level objectives. However, the presence of 51 respondents (32.9%) who utilise goal-oriented approaches indicates a growing interest in aligning system functionalities with overarching stakeholder goals, which is a perspective that can enhance requirement clarity and traceability.

4.2.7.4 Relevance to Visualising Textual User Stories

The prevalence of Use-Case Diagrams directly underscores the importance of this research, which aims to translate textual user stories into formalised UML models. By systematically mapping narrative requirements to use cases, practitioners can ensure that stakeholder perspectives are accurately captured and traceable throughout the development lifecycle. Additionally, the varied use of supporting diagrams, such as Class and Sequence Diagrams, demonstrates the multifaceted nature of system modelling, in which different diagram types collectively offer a holistic view of the system.

4.2.7.5 Significance for This Study

These findings highlight the need for a robust framework that can guide students and practitioners in transitioning from narrative, user-centric descriptions like textual user stories to a cohesive set of UML and complementary diagrams. Such a framework would streamline the modelling process and enhance the clarity and maintainability of requirements documentation. Ultimately, this research can contribute to more efficient, stakeholder-aligned software design practices by bridging the gap between textual user stories and conceptual diagrams.

4.3 Conclusion

This chapter has comprehensively explored the practices, preferences, and challenges associated with requirements gathering and conceptual diagram creation, as reflected in the survey data. Through an examination of various aspects ranging from the software development models in use, to the methods of requirement elicitation, documentation challenges, and diagramming approaches- a nuanced picture emerges of how students and practitioners currently navigate the transition from narrative textual user stories to structured system models.

A key finding is the dominance of Scrum and other Agile methodologies, which emphasises iterative development, frequent stakeholder feedback, and flexibility. This prevalence underscores the centrality of textual user stories in capturing functional requirements, given that they naturally align with Agile principles. Nevertheless, respondents highlighted a range of difficulties in effectively gathering and refining these textual user stories, including vague or evolving requirements, communication barriers, and

limited domain knowledge. These issues complicate the process of translating narrative requirements into precise design artefacts, such as UML use case diagrams.

The survey also shed light on the documentation methods employed after gathering requirements, revealing a strong preference for UML diagrams, particularly use-case, class, sequence, and activity diagrams. While these notations provide a structured approach to visualising system interactions and behaviours, participants reported challenges in selecting the most appropriate diagramming techniques and dealing with high time consumption in the documentation process. Additionally, many respondents noted that communication gaps between requirements engineers and development teams can lead to discrepancies and rework, further underscoring the importance of clear, consistent documentation practices.

Another critical dimension explored in this chapter is the problems encountered during requirement documentation, where high time consumption (77.6%) and confusion over suitable methods (57.8%) emerged as predominant concerns. These issues, combined with a lack of change management options for evolving requirements, underscore the need for more efficient and adaptive documentation workflows. The research also highlights the underutilisation of certain diagrammatic approaches, such as goal modelling, which could offer valuable insights into aligning textual user stories with higher-level project objectives.

In terms of conceptual diagram creation, the data indicate that most respondents (63.9%) rely on manual methods, while 36.1% have adopted semi-automated approaches. Manual diagramming grants a high degree of control and flexibility, yet it can be time-intensive and prone to errors. Semi-automated tools, on the other hand, promise efficiency and consistency but may require specialised training or additional software resources. This finding is particularly pertinent to the research objective of bridging textual user stories and

UML models, as it highlights both the opportunities and obstacles inherent in automating portions of the modelling process.

Overall, the results presented in this chapter underscore the significance of developing a robust framework or methodology to support the transition from textual user stories to formalised visual representations. Such a framework would ideally address the documented challenges namely, communication barriers, evolving requirements, and time constraints while harnessing the strengths of manual flexibility and the efficiency of semi-automated approaches. By offering clearer guidelines on how to structure textual user stories and map them systematically to UML diagrams, students and practitioners can achieve greater clarity, maintainability, and stakeholder alignment throughout the software development lifecycle.

In conclusion, the insights derived from this chapter provide a solid foundation for the subsequent design and validation of a systematic approach to user-story visualisation. The next chapter will build upon these findings to propose a methodology that aims to resolve the identified pain points, thereby enhancing both the educational experience for students and the practical outcomes of real-world software engineering projects. By addressing the existing gaps in requirement elicitation and documentation practices, this research seeks to facilitate more effective communication, reduce ambiguity, and ultimately improve the quality and reliability of the final software products.

CHAPTER 5

IMPLEMENTATION OF TEXTUAL USER STORIES TO UML USE CASE DIAGRAM FRAMEWORK

5.1 Overview

The purpose of this chapter is to present the detailed implementation of the Textual user stories to UML Use Case Diagram (US2UCD) framework. The US2UCD framework was conceived to address the limitations found in previous studies on automated and semi-automated transformation of requirements into UML models. Specifically, the framework aims to bridge the gap between structured textual user stories often used in Agile methodologies and UML use case diagrams, a standard notation widely employed in requirement engineering and systems analysis.

This chapter is divided into three major sections. The first section, Section 5.1, provides an overview of the framework's objectives, theoretical underpinnings, and high-level design. The second section, Section 5.2, describes the four main phases of the US2UCD framework which are requirements gathering, natural language processing, application of logical rules, and UML use case diagram generation. Each phase is explained in detail, including the tasks performed, the rationale behind them, and the resulting outputs. Finally, the last chapter conclude the implementation of the US2UCD framework. By the end of this chapter, readers will have a clear understanding of how textual user stories are processed and transformed into a UML use case diagram, as well as the benefits and potential challenges of the proposed framework.

5.2 US2UCD framework

The US2UCD framework is conceived as a semi-automated, model-driven approach for translating structured textual user stories into a UML use case diagram. This transformation is motivated by the need to bridge the gap between Agile requirement practices which often rely on concise, informal textual user stories and formal UML modelling, which is essential for clear communication, documentation, and design validation in software engineering. By integrating NLP with a set of logical rules, US2UCD ensures that the textual requirements are systematically analysed and then converted into UML elements that represent the system's functional scope.

Figure 5.1 illustrates the high-level pipeline of the US2UCD framework, divided into four main phases: Phase 1: Requirements Gathering, Phase 2: Natural Language Processing, Phase 3: Application of Logical Rules, and Phase 4: UML Use Case Diagram Generation.

Each phase addresses a critical step in progressively refining the raw textual data from textual user stories into a formal model. This structured approach helps ensure traceability of each UML element, which are actors, use cases, and relationships, can be traced back to the specific user story or stories that inspired it. Furthermore, by automating significant parts of the transformation process, the framework minimises human error, reduces the time and effort required for manual modelling, and promotes consistency across large sets of textual user stories.

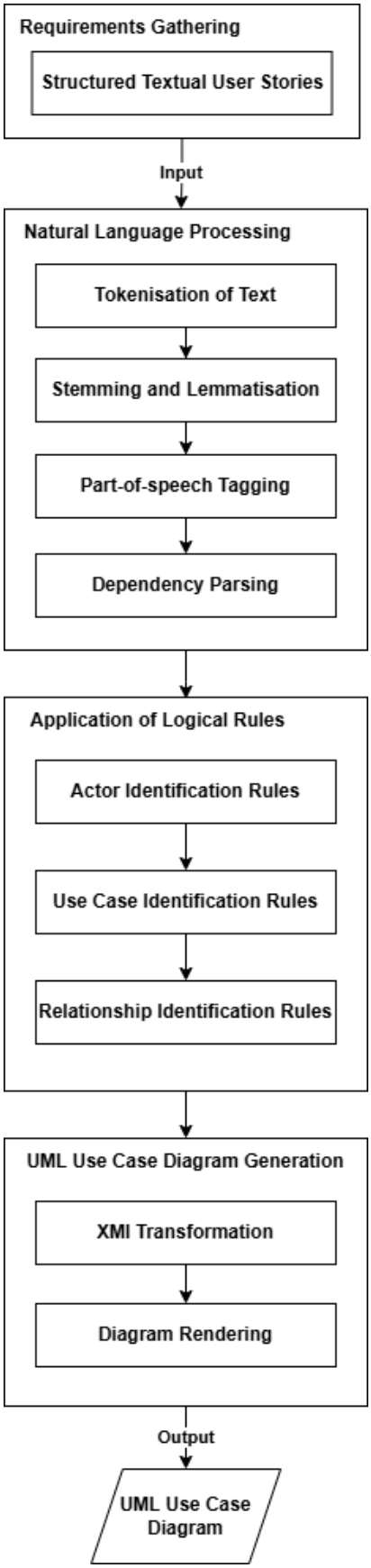


Figure 5.1: High-level pipeline proposed for US2UCD framework

The US2UCD framework, illustrated in Figure 5.1, presents a systematic, semi-automated framework for transforming structured textual user stories into UML use case diagrams. The framework consists of four sequential phases. Phase 1, Requirements Gathering, involves stakeholders providing requirements through a predefined template of textual user stories that captures essential UML elements, including actors, use cases, and relationships such as include, extend, and generalisation. This structured input ensures clarity and consistency, forming a reliable basis for subsequent automated processing.

In Phase 2, Natural Language Processing (NLP), the textual user stories are analysed using Stanza, a Python implementation of Stanford CoreNLP. Techniques such as tokenisation, lemmatisation, part-of-speech tagging, and dependency parsing convert raw text into linguistically annotated data, enabling the identification of subjects, actions, and modifiers and supporting the accurate extraction of UML elements.

Phase 3, Application of Logical Rules, interprets the annotated data using a set of rule-based heuristics. Noun phrases tagged as subjects are mapped to actors, while verb phrases representing functional goals are mapped to use cases. Specific syntactic patterns, such as “will cause” or “require,” trigger include relationships, whereas conditional clauses like “occurs if” signal extend relationships. This systematic mapping ensures that each UML component is consistently linked to the original user stories.

Phase 4, UML Use Case Diagram Generation, converts the structured specification into a visual model. Using PlantUML, the framework generates a text-based script defining actors, use cases, and their relationships, which is then rendered as a standard UML use case diagram. The resulting diagrams are clear, standardised, and traceable, facilitating validation, refinement, and communication among stakeholders.

Overall, the US2UCD framework integrates structured input, NLP analysis, rule-based interpretation, and automated visualisation to enable efficient and accurate transformation of user stories into UML models. This approach reduces manual effort, enhances consistency, and supports effective communication in the software development process.

5.2.1 Semi-Automated Process

While the US2UCD framework automates a substantial portion of the transformation from textual user stories to UML, it is semi-automated in nature:

i. Automation: Repetitive tasks such as text parsing, part-of-speech tagging, and straightforward actor/use case identification are handled by machine-driven NLP and rule application.

ii. Human-in-the-Loop: Domain experts and requirements engineers review the generated UML use case diagram, refining or validating the relationships, removing false positives such as irrelevant nouns tagged as actors, and resolving ambiguous cases. This iterative validation step ensures that the final diagram accurately reflects the real-world requirements.

5.2.2 Benefits of the US2UCD Approach

The benefits of US2UCD approach are listed as follows:

i. Consistency and Accuracy

By employing a standardised user story format and uniform NLP techniques, US2UCD reduces the likelihood of missing or misidentified actors and use cases. In the US2UCD framework, consistency and accuracy are measured through a multi-faceted evaluation approach that combines quantitative metrics, empirical

validation, and expert oversight. Consistency is assessed by evaluating the uniform application of the structured user story template and the reproducible mapping of similar linguistic patterns to corresponding UML elements across multiple case studies. This ensures that the framework produces reliably similar outputs for equivalent inputs, reinforcing standardization in the modelling process. Accuracy, on the other hand, is quantitatively measured using precision, recall, F1-score, and completeness metrics, which compare the automatically generated UML diagrams against manually constructed reference diagrams. These metrics quantify the correctness of extracted actors, use cases, and relationships by calculating true positives, false positives, and false negatives relative to the ground truth. The framework's performance is empirically validated through extensive case study testing across diverse domains, where each generated diagram is reviewed by domain experts who refine ambiguous mappings and remove false positives. Additionally, traceability is maintained by linking each diagram element back to its originating user story, enabling systematic verification and impact analysis. This comprehensive measurement strategy ensures that the US2UCD framework not only produces consistent and accurate UML use case diagrams but also remains adaptable and trustworthy for practical application in requirements engineering.

ii. Improved Communication

Translating textual user stories into a visual model helps align technical teams (developers, architects) with non-technical stakeholders (product owners, end users), facilitating better understanding of requirements.

iii. Reduced Manual Effort

Instead of manually constructing UML diagrams from large sets of textual user stories, requirements engineers can rely on automated parsing and rule-based identification, speeding up the modelling process.

iv. Enhanced Traceability

Each actor or use case can be traced back to the corresponding user story. This is particularly useful for impact analysis when requirements change.

v. Domain Independence

Although domain-specific rule sets can be added to improve accuracy, the core NLP-based pipeline is broadly applicable to textual user stories across diverse project domains.

5.2.3 Potential Limitations

i. Quality of Input: The accuracy of the framework depends on the clarity and completeness of the textual user stories. Poorly written, highly ambiguous, or inconsistent stories may yield suboptimal diagrams.

ii. Maintenance of Rules: As the domain evolves or new project-specific terminology emerges, the logical rules may require continuous updating.

iii. Partial Automation: Complex or specialised relationships might still require manual interpretation and refinement by domain experts.

In summary, US2UCD provides a systematic, semi-automated means of transforming structured textual user stories into a UML use case diagram. By consolidating NLP and logical inference within a single framework, it not only streamlines the modelling process but also enhances the quality, traceability, and maintainability of the resulting

artifacts. The subsequent sections delve into the technical and operational details of each phase, demonstrating how the framework converts the initial textual requirements into a final UML use case diagram.

5.2.4 Requirements Gathering

Requirements gathering forms the foundation of the US2UCD framework. In this phase, the goal is to collect textual user stories in a structured and consistent manner so that subsequent NLP and logical rule applications can be applied effectively.

5.2.4.1 Importance of Structured Requirements gathering

Structured requirements gathering is a critical step in ensuring that textual user stories are well-defined, unambiguous, and systematically mapped to UML elements. This approach enhances consistency, reduces errors in transformation processes, and aligns the extracted requirements with the final UML use case diagram.

i. Crucial Component of the Research
   a. The quality of the requirements directly influences the accuracy and reliability of the generated UML use case diagrams.
   b. Ambiguities or inconsistencies in textual user stories can propagate errors through all subsequent phases (NLP, rule application, diagram generation).

ii. Leveraging the 3Ws (Who, What, Why)
   a. A simplified yet powerful guideline for textual user stories is to identify **Who** is performing an action, **What** action they want to perform, and **Why** they want to perform it.
   b. This approach aligns with the widely used template popularised in Agile environments (Cohn, 2004):

"As a [type of user], I want [some goal], so that [some reason]."

c. By focusing on these three questions, stakeholders can more clearly articulate the purpose and benefits of each requirement.

iii. Structured Approach Using Text Templates

a. Employing a text template ensures consistency across textual user stories and reduces misinterpretation during NLP processing.

b. A structured template also encourages teams to consider potential relationships, such as include, extend, or generalisations of actors, at an early stage, thus minimising rework later.

5.2.4.2 Proposed Textual User Stories Template

The US2UCD framework adapts a structured user story template that explicitly maps user requirements to UML use case elements, addressing the limitations of informal natural language. This template enhances the conventional format by incorporating essential relationships, actor classifications, and dependencies required for UML diagram generation. The proposed user story template provided in Figure 5.2 aims to capture detailed information about system behaviour from the user's perspective.

As a (1) _____, I want to (2) _____. (1) _____ is a/may be a type of (4) _____. (2) _____ will cause/require (3) _____. (5) _____ occurs/present if (1) _____ (6) _____.

Figure 5.2: Proposed user story template

From the template, each numbering refers to details represented inside Table 5.1. In addition, Table 5.2 provides more details on each of the elements highlighted within the proposed textual user stories template.

Table 5.1: Reference for proposed textual user stories template

No	Information Gathered
1	Actor
2	Use case
3	Use case with include relationship
4	Generalisation of actor
5	Use case with extend relationship
6	Condition for extend relationship

Table 5.2: Elements of use case diagram depicted from the proposed textual user stories template.

Element	Template Format	Description
Actor Identification	"As a (1) [actor], I want to (2) [use-case]."	Defines the role or type of user interacting with the system, such as "customer", "admin", or "staff member."
Use Case Definition	"(2) [use-case] will cause/require (3) [use-case with an include relationship]."	Specifies the main functionality or goal that the actor intends to achieve.
Use Case with Include Relationship	"(3) [use-case with an include relationship]."	Represents a secondary or mandatory use case always invoked within the primary use case, such as

Table 5.2 continued

		"Placing an order" including "Selecting a payment method."
Generalisation of Actor	"(1) [actor] is a/may be a type of (4) [generalised actor]."	Defines hierarchical relationships among actors, such as "Premium customer" being a specialised type of "Customer."
Use Case with Extend Relationship	"(5) [use-case with an extend relationship] occurs/present if (6) [condition for extend relationship between use-cases]."	Captures optional functionality that extends the main use case under specific conditions, such as "Apply discount" extending "Place an order" if the customer is a premium member.
Condition for Extend Relationship Between Use Cases	"(6) [condition for extend relationship between use-cases]."	Identifies the condition under which the extended use case is triggered.

The proposed textual user stories template within the US2UCD framework constitutes a substantial enhancement relative to the original user story format by structurally augmenting narrative requirements to facilitate automated UML extraction. Whereas conventional user stories typically adhere to a high-level, narrative form such as "As a [role], I want to [action], so that [benefit]," the refined template systematically integrates explicit

indicators for UML-specific elements, encompassing actors, use cases, include and extend relationships, and generalisation hierarchies. This is accomplished through a structured syntax that not only records the actor and primary action but also necessitates the specification of dependency relationships, for example “will cause/require,” and conditional extensions, such as “occurs if,” which are frequently implied or omitted in traditional user stories. Consequently, the template mitigates much of the ambiguity and interpretive variability inherent in natural language, furnishing a consistent, machine-readable format that aligns directly with the constructs of UML use case diagrams. This enhancement therefore enables more precise natural language processing and rule-based mapping, reducing the need for manual inference during diagram generation. At the same time, it also enhances the traceability and completeness of the resultant models. These improvements address the limitations associated with informal requirement specifications in agile and model-driven development contexts.

By prompting stakeholders to consider these additional structured elements, the template encourages early identification of relationships and fosters clarity in actor-use case interactions. This structured approach aligns with best practices in use case–driven development and enhances traceability throughout the project lifecycle.

5.2.4.3 Eliciting Actors, Use Cases, and Relationships

This phase focuses on systematically identifying actors, use cases, and their relationships within the system. By applying structured analysis to textual user stories, actors and their interactions are extracted, distinguishing core functionalities from dependent or conditional behaviours. The identification of include, extend, and generalisation relationships ensures a comprehensive mapping of system functionalities, providing a strong foundation for UML use case diagram generation.

i. Identifying Actors

The process starts by listing the different roles or user types who will interact with the system. The template's "(1) _____ is a/may be type of (4) _____" prompts teams to note actor generalisations. For example, the term "student" and "instructor" both being sub-types of "user".

ii. Detailing Use Cases

Each user story's action (the "What") becomes a candidate use case. The structured format highlights primary goals and any secondary actions needed (include relationships).

iii. Uncovering Relationships

The explicit references to "include", "extend", and actor generalisations guide teams to articulate functional dependencies. For instance, "(2) will cause/require (3)" may hint at an include relationship if the second action is always needed to complete the first. Next, "(5) occurs/present if (6)" may indicate an extend relationship, describing optional or conditional behaviour (such as a discount that only applies if a coupon code is entered).

iv. Focusing on Why

Although the "Why" is often implicit, it remains essential for understanding the value or business rationale behind each requirement. Besides. encouraging stakeholders to articulate "Why" ensures alignment with business goals and user needs.

5.2.4.4 Ensuring Requirement Quality

To maximise the effectiveness of this structured approach, it is critical to review the textual user stories for clarity, completeness, and correctness before moving on to the NLP

phase. Table 5.3 describes criteria of the textual user stories that required reviews before it will be processed with NLP.

Table 5.3: Criteria for reviewing textual user stories

No.	Criteria	Description
1	Clarity	Avoids jargon or ambiguous language, ensuring that each placeholder in the template is unambiguously filled.
2	Completeness	Ensures that all relevant actors, use cases, and relationships are captured.
3	Correctness	Validates that the specified relationships accurately represent conditional or optional behaviour rather than essential steps that should be included in the primary use case.

During this review, domain experts and stakeholders play a pivotal role in confirming the validity of each user story. This collaborative approach aligns with Agile values (Beck et al., 2021) by involving the customer or end-user in the requirement definition, ensuring that what is captured truly represents real-world needs. Figure 5.3 shows the pseudocode for the logical flow of requirements gathering.

```
Input:
    Stakeholders = Set of project stakeholders
    Template = Proposed structured textual user story template
    QualityCriteria = {Clarity, Completeness, Correctness}

Output:
    Valid_US = Set of validated structured textual user stories

Begin
    Initialise Valid_US as empty set

    For each stakeholder s in Stakeholders do
        structured_us ← Stakeholder enters user story using Template

        If structured_us is filled then
            Add structured_us to Candidate_US
        End If
    End For

    For each user_story us in Candidate_US do
        clarity_ok ← CheckClarity(us)
        completeness_ok ← CheckCompleteness(us)
        correctness_ok ← CheckCorrectness(us)

        If clarity_ok AND completeness_ok AND correctness_ok then
            Add us to Valid_US
        Else
            Return us to stakeholder for correction
        End If
    End For

    Return Valid_US
End
```

Figure 5.3: Pseudocode for requirements gathering of US2UCD

5.2.5 Natural Language Processing

The NLP phase is a crucial step in the US2UCD framework, bridging the gap between textual user stories and logical rule application for generating UML use case diagrams. Building on the well-structured textual user stories gathered previously, the NLP component applies various techniques such as tokenisation, Part-of-Speech (POS) tagging, lemmatisation, and dependency parsing to reveal the underlying grammatical and semantic relationships within each user story. By doing so, it systematically identifies potential actors (nouns), actions (verbs), and objects, thereby reducing ambiguity and streamlining subsequent rule-based inference. Figure 5.4 shows the NLP techniques and their sequence applied to US2UCD, while Figure 5.5 shows the logical flow of how the NLP techniques are applied.

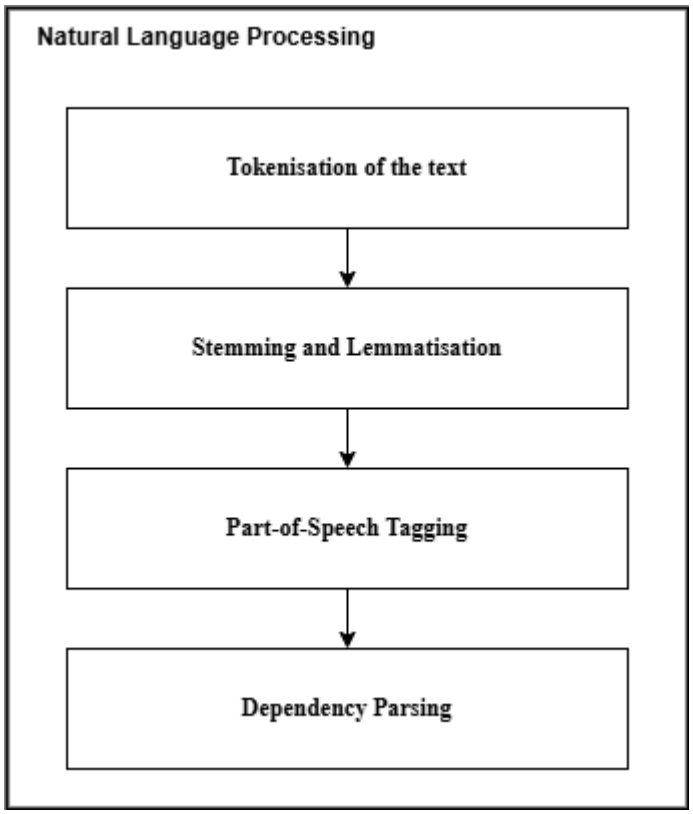


Figure 5.4: NLP techniques used in sequence for US2UCD

```
Input:
    US = Set of structured textual user stories
    D = Domain-specific dictionary
    SW = Refined stop word list
Output:
    NLP_Output = Set of linguistically annotated user stories
Begin
    Initialise NLP_Output as empty set
    For each user_story us in US do
        tokens ← Tokenise
        For each token t in tokens do
            if t not in SW then
                stem_t ← Stem(t)
                lemma_t ← Lemmatise(t)
            end if
        End For
        pos_tags ← POSTag(tokens)
        dep_graph ← DependencyParse (tokens, pos_tags)
        For each token t in tokens do
            if t exists in D then
                Override POS tag of t using D
            end if
        End For
        actors ← Extract nouns with subject relations from dep_graph
        actions ← Extract verbs from pos_tags
        objects ← Extract nouns with object relations from dep_graph

        annotated_us ← {
            Tokens: tokens,
            POS_Tags: pos_tags,
            Lemmas: lemma_t,
            Dependencies: dep_graph,
            Actors: actors,
            Actions: actions,
            Objects: objects
        }

        Add annotated_us to NLP_Output
    End For

    Return NLP_Output
End
```

Figure 5.5: Pseudocode for the implementation of NLP techniques in US2UCD

5.2.5.1 NLP Pipeline Components

The NLP phase typically comprises four main components, each contributing to a deeper understanding of the user story text:

i. Tokenisation

Tokenisation is a fundamental step in NLP, dividing textual content, such as user stories, into smaller units called tokens. Tokens typically encompass individual words, punctuation marks, numbers, and other symbols, each treated separately for detailed analysis. As an example, if the customised textual user stories are used as per below:

"As a (1) customer, I want to (2) place an order. (1) customer is a type of (4) user. (2) place an order will cause (3) process payment. (5) apply discount occurs if (1) customer (6) has a valid coupon."

After the tokenisation process has been done, the tokens would yield the sequence as:

["As", "a", "customer", ",", "I", "want", "to", "place", "an", "order", ".", "customer", "is", "a", "type", "of", "user", ".", "place", "an", "order", "will", "cause", "process", "payment", ".", "apply", "discount", "occurs", "if", "customer", "has", "a", "valid", "coupon", "."].

By breaking a sentence into discrete elements, tokenisation provides the groundwork for effective linguistic analysis. This initial preprocessing step is critical, as it directly impacts the performance of subsequent NLP processes, including stemming, lemmatisation, part-of-speech tagging, and dependency parsing. Accurate tokenisation ensures the clarity

and efficiency of downstream computational procedures, ultimately enhancing the reliability and precision of NLP applications.

ii. Stemming and Lemmatisation

Stemming and lemmatisation are essential linguistic processing techniques that are conducted following tokenisation. Stemming involves systematically removing common suffixes from words to produce simplified forms, transforming variations such as “placing”, “placed”, and “places” into the simpler root form “place”. In contrast, lemmatisation is a more sophisticated method that employs dictionaries along with grammatical and part-of-speech information to convert words accurately to their canonical or dictionary forms. For instance, the plural noun “customers” is standardised to its singular form “customer”. These processes play a vital role in standardising the textual data, effectively handling linguistic variations that might otherwise lead to fragmentation and inconsistencies. By reducing words to their essential forms, stemming and lemmatisation facilitate more reliable and precise identification of key elements within textual user stories, particularly actors represented by nouns and actions represented by verbs. Both stemming and lemmatisation are used in this research because they serve complementary purposes.

Stemming provides a fast and lightweight normalisation mechanism that reduces morphological variations early in the pipeline which improve processing efficiency and recall during pattern matching. Lemmatisation on the other hand, ensures linguistic correctness by producing valid dictionary forms, which is crucial for accurate actor and action identification when applying logical mapping rules. By combining both techniques, the framework achieves a balance between computational efficiency and semantic accuracy, ensuring robust and consistent extraction of conceptual model elements from textual user

stories. Consequently, these techniques contribute significantly to enhancing the quality and accuracy of subsequent NLP analyses, ensuring coherent and consistent interpretations throughout the processing pipeline.

iii. Part-of-Speech (POS) Tagging

Part-of-speech (POS) tagging assigns grammatical categories, such as nouns, verbs, adjectives, and adverbs, to each token obtained through tokenisation. Popular NLP libraries, including NLTK, spaCy, and Stanford CoreNLP, offer robust implementations of POS tagging that utilise statistical models or rule-based methods to achieve highly accurate results. In this research, Stanza, which is a Python version of Stanford CoreNLP, is used to perform the POS tagging. Assigning POS tags is crucial because it allows for a clear distinction between actors, typically represented by nouns, and actions, typically represented by verbs. Furthermore, POS tagging enables the filtering out of non-essential words such as conjunctions and determiners, thereby refining the dataset and ensuring the subsequent analysis of textual user stories remains precise and meaningful.

iv. Dependency Parsing

Dependency parsing is a core component of syntactic analysis in natural language processing. It involves examining the grammatical structure of a sentence to identify the relationships between individual words, particularly how they are hierarchically connected. This process enables the detection of dependencies, such as which words function as subjects, verbs, objects, and modifiers. By understanding these syntactic links, a more structured interpretation of sentence meaning can be achieved, which is essential for accurate

downstream processing. For example, in the sentence “As a customer, I want to place an order”, dependency parsing would reveal several important relationships:

a. “customer” is identified as the subject of the sentence, representing the actor performing the action.
b. “place” is recognised as the main verb, indicating the primary action within the sentence.
c. “order” is determined to be the object of the verb “place”, representing the entity being acted upon.

These relationships are crucial for the accurate interpretation of textual user stories within the proposed framework. Dependency relations serve as the structural basis for the application of logical rules. They enable the system to infer which nouns are associated with which verbs, which in turn allows the identification of actors and use cases. For instance, determining that “customer” is linked to the verb “place” supports the classification of “customer” as an actor and “place an order” as a corresponding use case. By leveraging dependency parsing, the framework is able to extract precise and contextually relevant information from structured textual user stories. This not only enhances the semantic clarity of the extracted data but also supports more accurate generation of UML use case elements, including actors, actions, and their interrelationships. Ultimately, dependency parsing plays a pivotal role in bridging the gap between natural language and formal modelling representations.

5.2.5.2 Handling Domain-Specific Vocabulary

Many software projects involve domain-specific terms such as SKU, invoicing, and premium membership that standard NLP tools may not recognise as significant. To improve accuracy:

i. Domain-Specific Dictionaries

   a. A custom dictionary can be incorporated to ensure that specialised terms are accurately tokenised and tagged.
   b. For instance, if “SKU” is always used as a noun, it should be annotated accordingly to avoid being misclassified.

ii. Stop Word Refinement

   a. Default stop word lists such as “the”, “of”, and “and” may be adapted to exclude or include certain words relevant to the domain.
   b. For instance, a term like “order” is crucial in an e-commerce context and must not be removed as a stop word.

iii. Iterative Refinement

   a. As textual user stories evolve or new terminology emerges, the NLP pipeline can be iteratively updated to reflect these changes.
   b. Collaboration with domain experts is essential to identify and validate domain-specific terms.

5.2.5.2 Integration with Structured Textual user stories

Because the US2UCD framework encourages a structured format for textual user stories the NLP phase benefits from clearer, more consistent text:

i. Reduced Ambiguity: The template ensures textual user stories follow a common pattern (*Actor → Action → Reason/Condition*).
ii. Focused Parsing: Since each user story is concise, the NLP tools can more effectively pinpoint relevant elements without being overwhelmed by extraneous text.
iii. Pre-Labelled Clues: The placeholders in the template ("(1) _____ is a/may be type of (4) _____") often contain implicit labelling of actors and relationships, which the NLP pipeline can exploit for higher precision.

5.2.5.3 Challenges and Mitigation Strategies

Despite the structured nature of textual user stories, several challenges may arise:

i. Synonymy and Polysemy
   a. Words like "order" can mean different things such as a command, or a purchase in an e-commerce context
   b. Word-sense disambiguation (WSD) techniques or domain-specific rules can help clarify the intended meaning.

ii. Complex Sentence Structures
   a. While textual user stories are generally short, they can sometimes contain nested clauses or multiple actions.

c. Ensuring each action is captured may require advanced dependency parsing and sentence splitting.

iii. Grammar and Spelling Variations

a. Stakeholders may use informal language or typos.
b. Preprocessing steps like spell checking or grammar correction can mitigate such issues, although these might introduce additional complexity.

iv. Performance and Scalability

a. Large projects can involve hundreds of textual user stories.
b. Efficient NLP tools or batch processing can be employed to handle high volumes of text without significant performance degradation.

5.2.5.4 Outputs of the NLP Phase

Upon completion of the NLP phase, each user story is augmented with structured annotations:

i. Tokenised Representation: A list of tokens representing each word or phrase.
ii. POS Tags: Each token labelled as noun (NN), verb (VB), adjective (JJ)
iii. Stems or Lemmas: Normalised word forms that facilitate consistent identification of concepts.
iv. Dependency Relations: A graph-like structure detailing how tokens connect like subject, object, and modifier.

These outputs serve as input to the Application of Logical Rules phase, where they are systematically interpreted to identify actors, use cases, and relationships (including

include, extend, and generalisation). By transforming raw text into linguistically rich data, the NLP component ensures that the subsequent rule-based analysis operates on accurate, high-quality information, thereby enhancing the reliability of the final UML use case diagram.

5.2.5.5 Selection of NLP Tool

This research employs Stanza as the chosen NLP tool for extracting structured information from textual user stories. Stanza is a modern NLP library developed by the Stanford NLP Group and serves as the Python variant of the widely used Stanford CoreNLP toolkit. The decision to use Stanza was informed by two primary considerations which are:

i. Prevalence in Related Work

A review of prior studies identified Stanford CoreNLP as the most frequently utilised NLP tool. As observed previously during the literature review, it was used in 12 instances, whereas alternative tools such as spaCy (4 instances), MADA+TOKAN, Word2Vec, WordNet, Apache OpenNLP, DeepSRL, TreeTagger, and others were each used only once. The adoption of Stanza, as a variant of Stanford CoreNLP aligns this research with standard academic practices while offering a more accessible implementation.

ii. Language Familiarity and Ease of Integration

Unlike Stanford CoreNLP, which is implemented in Java, Stanza is written in Python. Given that this project is developed in Python, the use of Stanza allows for direct integration without requiring additional interfacing or external calls to

Java-based tools. This decision reflects the researcher's familiarity with the Python programming language and the practical advantage of maintaining consistency within the development environment.

Stanza offers a complete suite of NLP capabilities essential for this study, including tokenisation, lemmatisation, part-of-speech tagging, and dependency parsing. These features support accurate syntactic and semantic interpretation of textual user stories and enable the application of logical rules for identifying UML elements such as actors, use cases, and their relationships. In addition, Stanza's support for the Universal Dependencies framework ensures that its linguistic annotations are both consistent and linguistically robust, further justifying its suitability for this work.

5.2.5.6 Summary

In summary, the NLP phase is pivotal in the US2UCD framework, converting structured textual user stories into a machine-interpretable form. By combining tokenisation, stemming and lemmatisation, POS tagging, and dependency parsing, potentially enhanced by domain-specific dictionaries, the framework effectively uncovers actors, actions, and relationships hidden within natural language. This systematic approach significantly reduces ambiguity, improves consistency, and accelerates the transition from textual requirements to model-based artifacts. The outputs of this phase pave the way for the application of logical rules, in which the extracted linguistic details are translated into UML elements to generate the final use case diagram.

5.2.6 Application of Logical Rules

After completing the NLP phase, the US2UCD framework transitions into applying logical rules to the annotated textual user stories. This phase systematically interprets the linguistic information part-of-speech tags, dependency relations, and domain-specific cues to identify actors, use cases, and their relationships such as include, extend, generalisation. By codifying domain knowledge and best practices into rule sets, the framework ensures a structured and repeatable approach for deriving UML elements from textual requirements. Figure 5.6 shows the sequence and types of logical rules applied for the input after NLP.

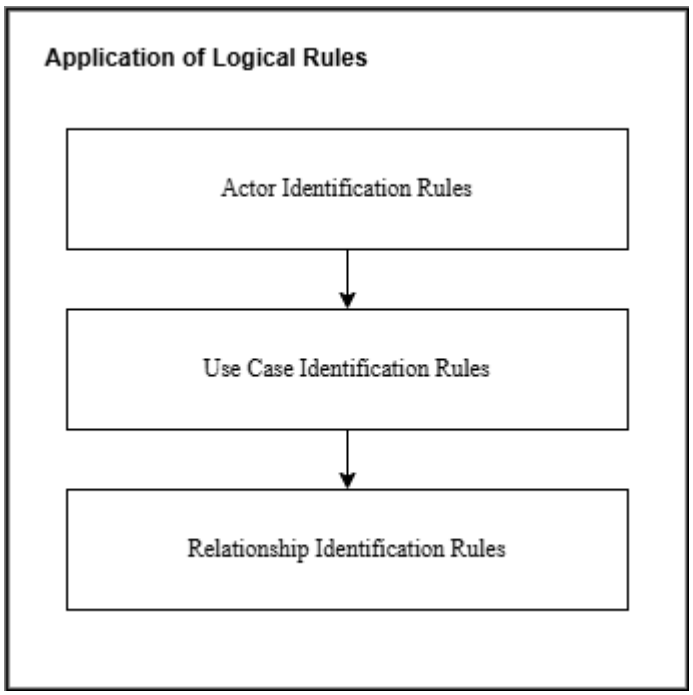


Figure 5.6: Types of Logical Rules applied for normalised input after NLP

5.2.6.1 Objectives of the Rule-Based Phase

The objective of the Rule-Based Phase is to translate the detailed linguistic annotations from the NLP phase into structured UML elements. This involves systematically applying logical rules to interpret part-of-speech tags, dependency relations, and domain-specific cues that enables the framework to accurately identify actors, use cases, and their relationships. This structured approach ensures that the translation from textual requirements to UML models is both consistent and repeatable.

The first step involves identifying actors within the system by analysing noun phrases in textual user stories. These nouns typically represent key entities interacting with the system, such as “customer”, “admin”, or “student”. Grammatical cues such as subject roles, aid in distinguishing actors, while domain-specific knowledge, including synonyms and role-based terminology, further refines the identification process.

Functional goals within textual user stories are extracted by identifying verb phrases that represent system actions. These actions define what the user seeks to accomplish, forming the basis of use cases. A distinction is made between primary actions, which serve as core use cases, and secondary or optional actions, which may indicate include or extend relationships.

Once actors and use cases are established, the relationships among them are determined. Associations between actors and use cases are identified, ensuring that each actor’s interactions with the system are accurately represented. Furthermore, interdependencies among use cases are analysed to infer include, extend, or generalisation relationships where applicable.

The final phase involves structuring the extracted information into a formal specification, typically in an intermediate format. This structured representation explicitly defines the identified actors, use cases, and their relationships. The resulting specification serves as the foundation for the automated generation of UML use case diagrams, ensuring a clear and standardised visual representation of system functionalities. Figure 5.7 shows the pseudocode for application of logical rules in US2UCD framework.

```
Begin
    Step 1: Initialise empty sets
        Actors
        UseCases
        Relationships

    Step 2: Identify actors
        For each user story do
            Extract noun phrases
            If noun phrase is subject of main verb
                Then add to Actors
            End if
        End for

    Step 3: Identify use cases
        For each user story do
            Extract main verb phrase
            If verb represents functional goal
                Then add to UseCases
            End if
        End for

    Step 4: Identify relationships
        For each user story do
            If actor performs use case
                Then create Association
            End if

            If story indicates mandatory sub-action
                Then create Include relationship
            End if

            If story indicates conditional action
                Then create Extend relationship
            End if
        End for

    Step 5: Identify generalisation
        Analyse actors and use cases across all stories
        If hierarchical pattern detected
            Then create Generalisation relationship
        End if

    Step 6: Generate structured model specification
        Output Actors, UseCases, Relationships
End
```

Figure 5.7: Pseudocode for application of logical rules

5.2.6.2 Rule Definition and Implementation

Logical rules can be implemented using various knowledge representation and inference techniques, such as:

i. If-Then Statements: Straightforward pattern-matching logic that checks for specific POS tags or dependency structures in the annotated user story.
ii. Production Rules: More sophisticated expert system approaches where each rule is triggered by certain conditions. As an example, if a noun phrase follows the phrase 'As a,' then classify it as an actor.
iii. Ontology or Domain Model: A formal domain ontology can help clarify relationships, synonyms, and hierarchical structures among actors and actions.

The specific rules employed in the US2UCD framework depend on:

i. Domain Constraints

   For instance, an e-commerce domain might require that "customer" and "seller" be recognised as distinct actors with certain common or overlapping use cases.

ii. Template Structure

   The structured user story format offers positional clues like "(1) _____ is a/may be type of (4) _____" that can directly guide rule creation.

iii. Project Requirements

   If the project needs to capture advanced relationships such as "extend" with specific conditions, the rule set can incorporate these triggers.

5.2.6.3 Actor Identification Rules

The actor identification rules define a structured approach to extracting actors from textual user stories using NLP. Noun phrase detection is applied to identify subjects of main verbs, classifying them as candidate actors. Role keywords and synonyms help recognise explicitly mentioned roles, mapping them to known actor categories using a domain dictionary. Additionally, actor generalisation rules establish hierarchical relationships between actors, ensuring that specialised roles inherit characteristics from broader actor categories. These rules enhance accuracy in UML use case diagram generation by systematically identifying and structuring actor relationships.

i. Noun Phrase Detection
    a. The NLP phase marks each token's part-of-speech such as *NN* and *NNS*, as well as dependency roles such as *subject*, *object*.
    b. A rule might state where if a token or phrase is the subject of the main verb in the user story, classify it as a candidate actor.
    c. For example: *"As a customer, I want to place an order."*. Actor produced is customer.

ii. Role Keywords and Synonyms
    a. Some textual user stories explicitly mention a role such as "As an admin…".
    b. Rules can map recognised role words like "admin", "manager", "premium member" to potential actors, possibly referencing a domain dictionary of known roles.

iii. Actor Generalisation

a. If a story states "(1) _____ is a/may be type of (4) _____," a rule can establish a generalisation hierarchy which can be in form of "premium customer" generalises to "customer".
b. This ensures the final UML model accurately reflects inheritance relationships among actors.

5.2.6.4 Use Case Identification Rules

The use case identification rules focus on extracting functional goals from textual user stories to accurately map them to UML use case diagrams. Verb phrase detection identifies the main action within a sentence, classifying it as a potential use case. Refining verbs into use cases ensures that only meaningful, high-level actions are considered, filtering out trivial or supporting actions that do not represent standalone functionalities. Additionally, mapping to the template helps determine include relationships by analysing dependencies within the structured user story format. These rules ensure precise extraction of use cases, maintaining clarity and consistency in UML model generation.

i. Verb Phrase Detection
    a. Use cases typically revolve around actions or functional goals.
    b. A rule might state if a token or phrase is the main verb in the sentence, consider it as a potential use case.
    c. For example: *"I want to place an order".* Producing use case: "Place an order."

ii. Refining Verbs into Use Cases
    a. Not every verb qualifies as a high-level use case; some are supporting or sub-actions.

b. Dependency or semantic rules can help filter out trivial actions such as "click", "open", "navigate" that may not represent a standalone use case.

iii. Mapping to the Template

a. If the structured template includes "(2) _____ will cause/require (3) _____," the rule can interpret these phrases as potential include relationships.

5.2.6.5 Relationship Identification Rules

The relationship identification rules define how actors and use cases interact within UML use case diagrams. Actor-use case association establishes a direct link when an actor initiates a use case. Include relationships to identify mandatory sub-use cases when a use case requires another action to be performed. Extend relationships capture optional behaviours that occur under specific conditions, ensuring flexibility in system modelling. Lastly, generalisation detects patterns across multiple textual user stories, identifying cases where actors or use cases share common attributes, leading to hierarchical structuring. These rules enhance the accuracy of UML relationship mapping and maintain logical consistency in the model.

i. Actor-Use Case Association

a. A basic association is established when an actor performs or initiates a use case.

b. A simple rule might say: "If a noun phrase (actor) is directly linked via a verb phrase (use case), create an association."

ii. Include Relationships

a. Include implies a mandatory or common sub-use case.

b. A rule may be: “If a user story states ‘(2) _____ will cause/require (3) _____,’ interpret (3) as an included use case.”

c. Example: “Placing an order” includes “selecting a payment method” if the user story explicitly mentions it as a required step.

iii. Extend Relationships

a. Extend indicates an optional or conditional use case.

b. The structured template might have “(5) _____ occurs/present if (6) _____,” suggesting an extend scenario.

c. Example: “Applying a discount” extends “placing an order” if a coupon code is provided.

iv. Generalisation

a. In addition to actor generalisation, use cases can also exhibit inheritance which however is less common.

b. A rule could detect repeated patterns across multiple textual user stories, prompting a generalised or abstract use case.

5.2.6.6 Output of the Rule-Based Phase

Upon applying the rules, the system generates a model specification often in a tabular or XML-based format listing:

i. Actors: Derived from noun phrases and role references.

ii. Use Cases: Identified verbs or functional goals, often labelled with a short action phrase.

iii. Relationships:

c. Associations between actors and use cases.

d. Include or extend dependencies among use cases.

e. Generalisation hierarchies for actors or use cases.

This structured specification provides clear traceability from each actor and use case back to the original textual user stories. It also ensures the data is in a machine-readable format, ready for the UML Use Case Diagram Generation phase.

5.2.6.7 Example Scenario

To illustrate the application of the structured user story template in identifying actors, use cases, and relationships, consider the following user story:

"As a premium customer, I want to place an order. A premium customer is a type of customer. Placing an order will require selecting a payment method. Applying a discount occurs if the cart total exceeds RM100."

i. Actor Identification

The "premium customer" is identified as the primary actor interacting with the system. A generalisation relationship is established: "premium customer" is a specialised type of "customer" inheriting its attributes.

ii. Use Case Identification

"Place an order" is identified as the primary use case associated with the premium customer. Then, "Selecting a payment method" is recognised as a mandatory action within the process of placing an order.

iii. Relationship Identification

The relationships are identified based on their different types, which are:

a. Association: A direct link is created between “premium customer” and “place an order” to represent the actor's interaction with the system.
b. Include Relationship: Since “placing an order” requires “selecting a payment method”, the latter is included as a mandatory sub-use case.
c. Extend Relationship: “Applying a discount” is recognised as an optional extension of “placing an order” that occurs only if the cart total exceeds RM100.

iv. Final UML Specification

The UML specification produced after applying the rules would look like below:

a. Actor: Premium Customer (inherits from Customer)
b. Primary Use Case: Place an Order
c. Included Use Case: Selecting a Payment Method
d. Extended Use Case: Apply a Discount (extends Place an Order)
e. Condition for Extension: “Cart total exceeds RM100”

This structured approach ensures that the user story is systematically decomposed into UML elements, facilitating a seamless transition from textual requirements to UML use case diagrams while maintaining clarity and traceability.

5.2.6.8 Challenges and Best Practices

The rule-based approach to identifying actors, use cases, and relationships in textual user stories presents several challenges, requiring best practices to ensure accuracy and adaptability. Overfitting rules can lead to failures when applied to varied user story formats, necessitating iterative refinement and testing. Handling ambiguities is crucial, as certain terms may be misclassified without additional NLP checks or domain expertise. Maintaining domain knowledge ensures that rules remain relevant as systems evolve, requiring

collaboration with domain experts for validation. Lastly, integration with NLP demands high accuracy in part-of-speech tagging and dependency parsing, as errors in these processes can significantly impact the reliability of rule-based interpretations. Addressing these challenges through systematic refinement and evaluation enhances the effectiveness of UML use case extraction.

i. Overfitting Rules
   a. Rules that are overly specific to certain sentence structures may fail when applied to new or varied user story formats.
   b. Best Practice: Iterative refinement and validation using diverse datasets helps maintain robustness and generalizability (Maatuk & Abdelnabi, 2021). Regularly testing rules against a representative corpus of user stories prevents overfitting and supports adaptability.

ii. Handling Ambiguities
   a. Certain phrases, such as “administrator” versus “administrator account,” can be interpreted as either an actor or an object, leading to misclassification.
   b. Best Practice: Enhanced disambiguation through extended NLP analysis, such as dependency parsing and semantic role labelling, improves term classification (Schlutter & Vogelsang, 2020). Additionally, incorporating domain-specific dictionaries clarifies term usage in context (Alami et al., 2020).

iii. Maintaining Domain Knowledge
   a. As systems and domains evolve, rules must be updated to reflect new terminology, roles, and relationships.
   b. Best Practice: Continuous collaboration with domain experts ensures that rules remain relevant and accurately capture evolving requirements (Javed & Lin, 2018).

This participatory validation aligns with Agile principles and supports sustainable rule maintenance.

iv. Integration with NLP

a. High accuracy in part-of-speech tagging and dependency parsing is critical where errors in these stages propagate and affect rule-based outcomes.

b. Best Practice: Regular evaluation of NLP accuracy and the integration of domain-adapted lexicons and parsers mitigate parsing errors and improve input quality (Ferrari et al., 2022). Implementing validation checkpoints between NLP and rule application stages further reduces error propagation.

By systematically applying these best practices which includes iterative refinement, expert collaboration, enhanced disambiguation, and rigorous NLP validation, the framework enhances the reliability, accuracy, and adaptability of UML use case extraction from textual user stories.

5.2.6.9 Summary

In conclusion, the Application of Logical Rules phase translates annotated textual user stories into a structured model of actors, use cases, and relationships. By leveraging actor identification, use case extraction, and relationship inference rules, US2UCD systematically bridges the gap between textual requirements and formal UML elements. This approach not only reduces manual effort but also enhances consistency, traceability, and maintainability of the resulting models. The final output of this phase sets the stage for UML Use Case Diagram Generation phase where these logically derived elements are transformed into a visual UML representation.

5.2.7 Generation of UML use case diagram

After the logical rules have been applied to extract actors, use cases, and their interrelationships from the annotated textual user stories, the final phase of the US2UCD framework is the Generation of the UML Use Case Diagram. This phase converts the structured model specification into a standardised, visual representation for system analysis, validation, and stakeholder communication. Figure 5.8 shows the UML use case diagram generation phase.

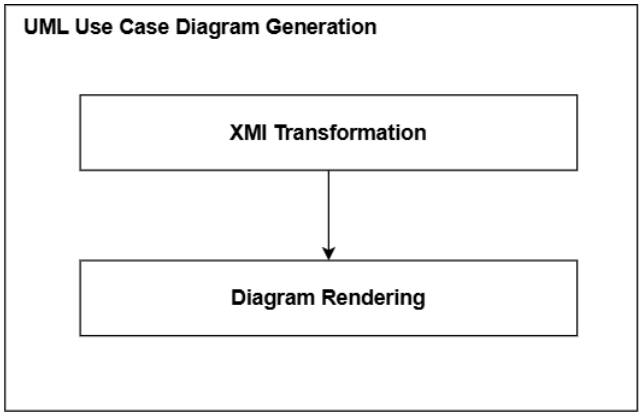


Figure 5.8: UML Diagram generation phase of US2UCD

5.2.7.2 Transformation Process

The generation of the UML use case diagram is achieved through a two-step process which are XMI transformation and diagram rendering and visualisation. The generation of UML use case diagrams in this study is achieved through a two-phase process which are the transformation of structured model data into a diagram specification format, and the subsequent visualisation of that specification using PlantUML.

The first phase involves converting the structured model, which consists of identified actors, use cases, and their associated relationships, into the textual syntax required by

PlantUML. Unlike approaches that rely on XMI, this study adopts PlantUML's concise scripting language as an intermediate format. Each element from the earlier stages of processing is mapped to its equivalent representation in PlantUML. For example, actors are declared as actor elements, use cases are specified as use case elements, and the relationships between them, including associations, inclusion, and extension, are defined using directional lines and standard UML annotations. This transformation is performed automatically through a set of custom scripts that interpret the extracted model components and construct a valid PlantUML script, ensuring that the structure of the textual user stories is preserved in the resulting diagram specification.

In the second phase, the generated PlantUML script is rendered into a visual diagram. This is carried out using either the PlantUML desktop tool or an online PlantUML rendering service. Upon execution, PlantUML reads the script and produces a graphical representation of the use case model. The resulting diagram visually presents actors as simplified human figures, use cases as elliptical nodes, and the relationships between them as clearly annotated connecting lines. The rendering process is fully automated, enabling rapid generation of visual models from text with minimal manual intervention.

Although the initial layout is produced automatically, slight manual refinements may be applied to improve clarity and presentation, such as adjusting the positioning of elements or refining the spacing between diagram components. These refinements help ensure that the final diagram is not only technically accurate but also easy to interpret by project stakeholders.

By using PlantUML as the visualisation tool, this study benefits from a lightweight, text-driven modelling approach that facilitates automation, version control, and easy

integration with documentation workflows. The chosen method supports both the structural rigour required for UML compliance and the flexibility needed for efficient implementation in a research setting. Last but not least, Figure 5.9 shows the snippet of pseudocode for XMI transformation while Figure 5.10 shows snippets of pseudocode for diagram rendering.

```
INITIALISE empty list of actors
INITIALISE empty list of use_cases
INITIALISE empty list of relationships

FOR EACH user_story IN user_stories DO
      EXTRACT actor FROM user_story
      ADD actor TO actors list IF not already present

      EXTRACT main use case FROM user_story
      ADD use case TO use_cases list IF not already present

      EXTRACT includes and extends FROM user_story
      ADD them TO relationships list

      EXTRACT generalisation relationships (actor inheritance)
      ADD to relationships list
END FOR
CREATE XMI model structure
FOR EACH actor, use case, and relationship
      ADD corresponding XMI elements
END FOR
```

Figure 5.9: Pseudocode for XMI transformation

```
INITIALISE empty string for PlantUML script
ADD "@startuml" TO script

FOR EACH actor DO
      ADD PlantUML actor declaration (e.g., actor "Passenger" as
passenger_id)
END FOR
FOR EACH use_case DO
      ADD PlantUML use case declaration (e.g., usecase "Search Buses" as
UC1)
END FOR
FOR EACH relationship DO
      IF relationship is actor -> use case association
            ADD arrow from actor to use case
      ELSE IF relationship is <<include>>
            ADD include arrow
      ELSE IF relationship is <<extend>>
            ADD extend arrow
            ADD condition note beside extension point
      ELSE IF relationship is generalisation
            ADD inheritance
END FOR
ADD "@enduml" TO script
SAVE the PlantUML script to a file
EXECUTE PlantUML with the .puml file to generate the visual diagram
OUTPUT the rendered diagram as an image
PRINT "UML use case diagram generated and visualised successfully using
PlantUML."
```

Figure 5.10: Pseudocode for diagram rendering

5.2.7.3 Implementation Considerations

The implementation of the diagram generation and visualisation process involved several important considerations, particularly in relation to tool compatibility, adaptability, accuracy, and the capacity to manage increasing complexity.

The choice of modelling tool had a significant influence on the development approach. This study employed PlantUML as the preferred tool for generating use case diagrams. PlantUML was selected for its ability to convert structured textual representations into standard UML diagrams without relying on XML-based exchange formats. Its support for widely accepted UML conventions and its straightforward text input method made it a practical and efficient solution for integrating diagram generation within the proposed

framework. By using PlantUML, the need for external graphical modelling software was removed, allowing diagrams to be produced consistently and programmatically.

Another consideration was the flexibility to adapt the transformation process to suit specific domain requirements. The mapping of structured textual user stories to diagram elements can be customised to reflect particular modelling conventions or rules. For instance, variations in how relationships are expressed within the textual user stories can be accounted for by modifying the logic used to represent inclusion, extension, or other connections between use cases. This adaptability ensures that the generated diagrams remain relevant and meaningful within the context of the system being described.

Accuracy in representation was maintained through the inclusion of error checking routines during the transformation process. The system verifies that all essential elements, such as actors and use cases, are properly extracted and represented in the diagram script. If any inconsistency is detected, such as a missing association or an undefined element, it is flagged for manual review. This precaution helps preserve the correctness and completeness of the final output.

Finally, the ability to scale the implementation for larger systems was taken into account. In projects involving a considerable number of textual user stories, the resulting diagrams can become increasingly complex. To address this, the system allows for structured arrangement of elements, applying layout strategies that enhance visual clarity. This approach ensures that even in more detailed models, the diagrams remain readable and informative.

Taken together, these considerations support the practical effectiveness of the implementation. The selected tools and techniques enable automated, adaptable, and

accurate generation of UML use case diagrams in a manner that is suitable for a variety of system sizes and complexities.

5.2.7.4 Benefits and Outcomes

The visual model significantly enhances clarity by providing stakeholders with an intuitive representation of system requirement which enables them to comprehend complex functionalities briefly. This improved visualisation fosters more effective communication, bridging the gap between developers and non-technical stakeholders by allowing them to quickly grasp system interactions. As a result, it facilitates smoother validation processes and supports iterative development by ensuring that feedback can be incorporated seamlessly. Furthermore, the model reinforces traceability by linking every diagram element to its originating user story. This structured approach guarantees that the model remains aligned with stakeholder requirements throughout the project lifecycle, reducing ambiguity and enhancing the overall coherence of the system design.

5.2.7.5 Summary

In summary, the Generation of the UML Use Case Diagram phase converts a rich, structured model specification into a formal, visual UML artifact. This phase leverages standard XMI transformation and automated rendering techniques, ensuring that the final diagram is both accurate and easy to interpret. The resulting diagram serves as a pivotal tool for system analysis, design validation, and ongoing communication between technical teams and stakeholders.

CHAPTER 6

RESULTS AND EVALUATION OF THE GENERATED DIAGRAMS FROM US2UCD

6.1 Chapter Overview

This chapter explains the results from using the system to generate UML use case diagrams. It starts with an experiment that uses a set of structured textual user stories to see how well the system can identify actors, use cases, and their relationships. The output from this experiment is then evaluated using a scoring methods, including precision, recall, F1 score, and completeness.

After that, the chapter shows an evaluation of the system based on several case studies. These case studies come from software engineering students who created textual user stories and diagrams for different systems. The system was tested on each of these samples to see how accurate and consistent the results were. By using both a controlled experiment and real examples, this chapter gives a full picture of how well the system works.

6.2 Results of US2UCD Implementation

This subsection describes the results of the implementation for US2UCD. An experiment has been conducted with a set of textual user stories that serve as test input for the model. Textual user stories are written in the form of a proposed template. Results from the experiment will also be presented at the end of this subsection.

6.2.1 Textual user stories sample for conducting the experiment

The first thing to be identified before starting to write the textual user stories is the domain to be used. In this experiment, the domain to be used is an online bus ticketing system. This domain was chosen because it is one of the most common types of system that

is available over the internet. Similar systems can be found in hotel booking systems, airplane booking systems, and many others. After the domain has been identified, a list of textual user stories has been written to replicate the real-world scenario for this kind of system. Table 6.1 records the list of textual user stories that have been used to conduct this experiment. These textual user stories will be fed to the US2UCD prototype to observe the generated use case diagram.

Table 6.1: List of textual user stories used to conduct this experiment

No.	User Story
1	As a passenger, I want to search for available buses. Passenger is a type of user. Searching for available buses requires entering travel details. Filtering options occur if the passenger selects specific preferences.
2	As a passenger, I want to book a bus ticket. Passenger is a type of user. Booking a bus ticket requires selecting a seat. Payment processing occurs if the passenger confirms the booking.
3	As a passenger, I want to cancel my bus ticket. Passenger is a type of user. Cancelling a bus ticket requires verifying the booking. Refund processing occurs if the cancellation meets the refund policy.
4	As a bus operator, I want to add new bus routes. Bus operator is a type of service provider. Adding new bus routes requires specifying the departure and destination. Route validation occurs if the system detects route duplication.
5	As a bus operator, I want to update bus schedules. Bus operator is a type of service provider. Updating bus schedules requires selecting the affected route. Passenger notifications occur if schedule changes impact existing bookings.
6	As an administrator, I want to manage passenger accounts. Administrator is a type of staff. Managing passenger accounts requires verifying identity. Account suspension occurs if the passenger violates terms of service.
7	As a passenger, I want to choose my preferred seat. Passenger is a type of user. Choosing a preferred seat requires viewing seat availability. Alternative suggestions occur if the preferred seat is already booked.

Table 6.1 continued

8	As a driver, I want to view my assigned trips. Driver is a type of service provider. Viewing assigned trips requires logging into the driver portal. Route changes occur if the bus operator updates the schedule.
9	As a passenger, I want to receive booking confirmation. Passenger is a type of user. Receiving booking confirmation requires completing the payment. Ticket issuance occurs if the transaction is successful.
10	As a system administrator, I want to generate sales reports. System administrator is a type of staff. Generating sales reports requires selecting a date range. Detailed analytics occur if the administrator applies advanced filters.

6.2.3 Generating use case diagram from the structured textual user stories

The textual user stories specified earlier have been used as input into the US2UCD prototype and results can be obtained. The results, including extracted actors, use case diagrams, and relationships, will be gathered and analysed in this subsection. Table 6.2 shows the extracted actor from the textual user stories list, followed by Table 6.3 which shows the use cases list. Table 6.4 listed included use cases and Table 6.5 shows extended use cases. Furthermore, PlantUML code have also been generated as part of the pipeline. The generated code is shown in Figure 6.1. Lastly, the generated use case diagram is shown in Figures 6.2, 6.3, and 6.4. For the generated diagrams, it is split into several parts to improve readability of the diagram itself since the diagram produced is in landscape format.

Table 6.2: Extracted use case diagram's actors

No.	Actor
1	Bus Operator
2	Driver
3	Passenger
4	System Administrator

Table 6.3: Extracted use case diagram's use cases

No.	Use Case
1	Search for available buses
2	Book a bus ticket
3	Cancel my bus ticket
4	Choose my preferred seat
5	Generate sales reports
6	Manage passenger accounts
7	Receive booking confirmation
8	Add new bus routes
9	Update bus schedules
10	View my assigned trips

Table 6.4: Extracted use case diagram's use cases with include relationship

No.	Primary Use Case	Included Use Case
1	Search for available buses	Entering travel details
2	Book a bus ticket	Selecting a seat
3	Cancel my bus ticket	Verifying the booking
4	Add new bus routes	Specifying the departure and destination
5	Update bus schedules	Selecting the affected route
6	Manage passenger accounts	Verifying identity
7	Choose my preferred seat	Viewing seat availability
8	View my assigned trips	Logging into driver portal
9	Receive booking confirmation	Completing the payment
10	Generate sales reports	Selecting a date range

Table 6.5: Extracted use case diagram's use cases with extend relationship

No.	Use Case	Extended Condition
1	Search for available buses	If the passenger selects specific preferences
2	Book a bus ticket	If the passenger confirms the booking
3	Cancel my bus ticket	If the cancellation meets the refund policy
4	Add new bus routes	If the system detects route duplication
5	Update bus schedules	If schedule changes impact existing bookings
6	Manage passenger accounts	If the passenger violates terms of service
7	Choose my preferred seat	If the preferred seat is already booked
8	View my assigned trips	If the bus operator updates the schedule
9	Receive booking confirmation	If the transaction is successful

Table 6.5 continued

10	Generate sales reports	If the administrator applies advanced filters

```
@startuml

actor "Bus Operator" as A1
actor "Driver" as A2
actor "Passenger" as A3
actor "System Administrator" as A4


usecase "Search for available buses" as UC1
usecase "Book a bus ticket" as UC2
usecase "Cancel my bus ticket" as UC3
usecase "Choose my preferred seat" as UC4
usecase "Generate sales reports" as UC5
usecase "Manage passenger accounts" as UC6
usecase "Receive booking confirmation" as UC7
usecase "Add new bus routes" as UC8
usecase "Update bus schedules" as UC9
usecase "View my assigned trips" as UC10


A1 --> UC8
A1 --> UC9
A2 --> UC10
A3 --> UC1
A3 --> UC2
A3 --> UC3
A3 --> UC4
A3 --> UC7
A4 --> UC5
A4 --> UC6


usecase "Entering travel details" as I1
usecase "Selecting a seat" as I2
usecase "Verifying the booking" as I3
usecase "Specifying the departure and destination" as I4
usecase "Selecting the affected route" as I5
usecase "Verifying identity" as I6
usecase "Viewing seat availability" as I7
usecase "Logging into driver portal" as I8
usecase "Completing the payment" as I9
usecase "Selecting a date range" as I10

UC1 ..> I1 : <<include>>
UC2 ..> I2 : <<include>>
UC3 ..> I3 : <<include>>
UC8 ..> I4 : <<include>>
UC9 ..> I5 : <<include>>
UC6 ..> I6 : <<include>>
UC4 ..> I7 : <<include>>
UC10 ..> I8 : <<include>>
UC7 ..> I9 : <<include>>
UC5 ..> I10 : <<include>>

usecase "Search for available buses (extended)" as EX1
UC1 -[#red,dashed]-> EX1 : <<extend>>

usecase "Book a bus ticket (extended)" as EX2
UC2 -[#red,dashed]-> EX2 : <<extend>>


usecase "Cancel my bus ticket (extended)" as EX3
UC3 -[#red,dashed]-> EX3 : <<extend>>

usecase "Add new bus routes (extended)" as EX4
UC8 -[#red,dashed]-> EX4 : <<extend>>

usecase "Update bus schedules (extended)" as EX5
UC9 -[#red,dashed]-> EX5 : <<extend>>

usecase "Manage passenger accounts (extended)" as EX6
UC6 -[#red,dashed]-> EX6 : <<extend>>

usecase "Choose my preferred seat (extended)" as EX7
UC4 -[#red,dashed]-> EX7 : <<extend>>

usecase "View my assigned trips (extended)" as EX8
UC10 -[#red,dashed]-> EX8 : <<extend>>

usecase "Receive booking confirmation (extended)" as EX9
UC7 -[#red,dashed]-> EX9 : <<extend>>

usecase "Generate sales reports (extended)" as EX10
UC5 -[#red,dashed]-> EX10 : <<extend>>

@enduml
```

Figure 6.1: PlantUML code generated by US2UCD

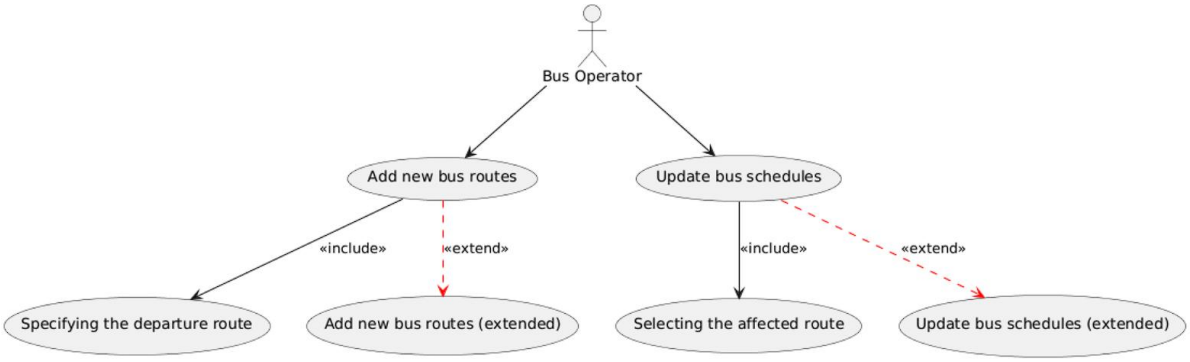


Figure 6.2: Generated use case diagram for Bus Operator

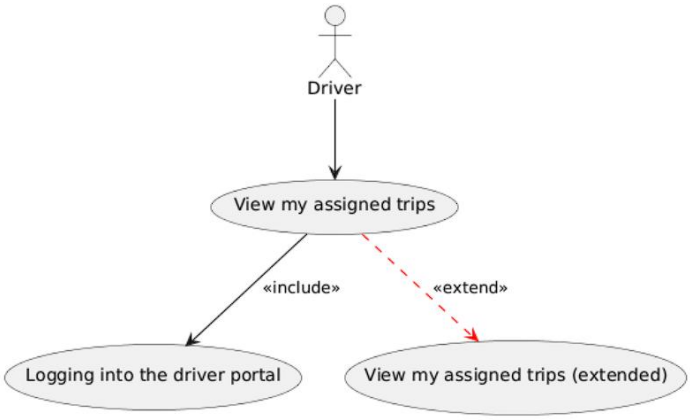


Figure 6.3: Generated use case diagram for Driver

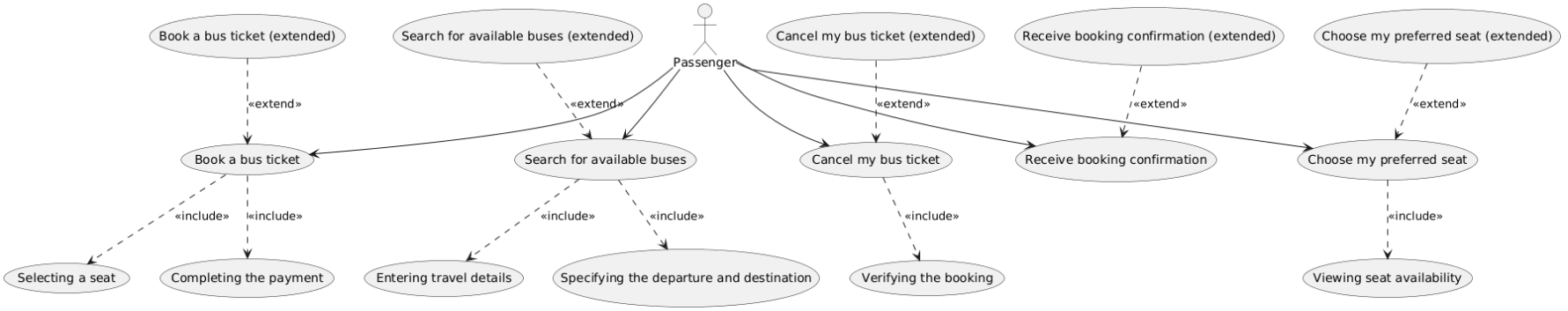


Figure 6.4: Generated use case diagram for Passenger

6.3 Empirical Evaluation of US2UCD

This subsection presents the detailed evaluation of the proposed framework which is US2UCD, focusing on its ability to accurately generate UML use case diagram from structured textual user stories. Metrics such as precision, recall, and usability scores were applied to assess the system's effectiveness. Details of the evaluation methods will be discussed in the following subsections.

6.3.1 Evaluation methods

To comprehensively evaluate the performance and accuracy of the proposed US2UCD approach, a set of quantitative and qualitative metrics was utilised. In this context, performance refers to the operational effectiveness of the US2UCD system in transforming structured textual user stories into UML use case diagrams, including reliability, consistency, and usability of the generated outputs under varying input conditions. It encompasses measurable aspects such as scalability, system reliability, and the usability of the generated outputs, reflecting how well the system operates under varying conditions and data volumes. Accuracy, on the other hand, denotes the degree to which the generated UML elements including actors, use cases, and relationships, correctly correspond to the intended requirements specified in the textual user stories. It is quantitatively assessed by comparing the system's outputs against a manually validated reference model, focusing on the precision, recall, and completeness of the identified diagrammatic elements. Together, these metrics provide a comprehensive assessment of both the functional correctness and operational effectiveness of the framework in transforming narrative requirements into formal visual models. These methods aim to determine how effectively the system generates UML use case diagrams from structured textual user stories. The evaluation not only focuses on the correctness of the elements generated but also on the completeness, usability, and overall

user experience of the system. This section provides a detailed explanation of the metrics and methodologies used for evaluation.

6.3.2 Quantitative Metrics

Quantitative metrics play a crucial role in objectively assessing the technical performance of the US2UCD. The primary metrics employed in this study include precision, recall, F-1 score, and completion score.

i. Precision

Precision measures the system's ability to correctly identify relevant elements in the UML use case diagram. It focuses on minimising false positives by ensuring that all elements generated by the system are meaningful and relevant to the input textual user stories. Precision is calculated using the following formula in Equation 6.1.

Equation 6.1

$$Precision = \frac{True\ Positive\ (TP)}{True\ Positive\ (TP) + False\ Positive\ (FP)}$$

A high precision score indicates that the system avoids generating extraneous or incorrect diagram components, which is critical for producing clear and accurate diagrams.

ii. Recall

Recall evaluates the system's capability to identify all relevant elements present in the ground-truth UML use case diagram. This metric ensures that the system captures the full range of necessary components, thereby minimising omissions. Recall is defined in Equation 6.2.

$$Recall = \frac{True\ Positive\ (TP)}{True\ Positive\ (TP) + False\ Negative\ (FN)}$$ Equation 6.2

A high recall score signifies that the system performs well in generating comprehensive diagrams that reflect the entirety of the input textual user stories.

iii. F-1 Score

While precision and recall individually measure specific aspects of performance, the F-1 score combines these metrics into a single value to assess the balance between them. It is particularly useful when there is a trade-off between avoiding false positives (precision) and capturing all relevant elements (recall). The F-1 score is calculated Equation 6.3.

$$F1 = \frac{2\ X\ Precision\ X\ Recall}{Precision + Recall}$$ Equation 6.3

A high F-1 score reflects the system's overall effectiveness in generating accurate and complete diagrams.

iv. Completion Score

The completion score measures the alignment between the generated use case diagram and the manually created reference diagram in terms of both structure and semantics. This score accounts for:

a. Structural accuracy: Matching the number of actors, use cases, and relationships.
b. Semantic accuracy: Ensuring the naming and relationships of elements align with the intent of the textual user stories.

c. Weighted scoring is applied to evaluate how completely the generated diagram represents the input textual user stories.

6.4 Evaluation of US2UCD generated diagram

The evaluation of the generated UML use case diagrams was carried out by comparing them with a manually constructed reference model derived from ten structured textual user stories. The reference model was created by domain knowledgeable participants who interpreted the user stories and modelled the diagrams based on their understanding of the requirements. This manually constructed model served as the ground truth for evaluation. The comparison focused on four categories of elements, namely actors, primary use cases, include relationships, and extend relationships. Each category was assessed using precision, recall, F1 score, and completeness metrics.

6.4.1 Generated Actors

The expected actors were Passenger, Bus Operator, Driver, Administrator, and System Administrator. Only three of these were generated correctly, namely Passenger, Bus Operator, and Driver. The Administrator and System Administrator were not represented, although both were clearly defined in the original textual user stories. However, there is no incorrect actors were included in the diagram. The evaluation score for actors is shown in Table 6.5.

Table 6.5: Evaluation score for actors

No	Evaluation Method	Score
1	Precision	1
2	Recall	0.6
3	F1 score	0.75
4	Completeness	0.6

The System Administrator actor was not shown visually, despite being correctly included in the PlantUML code. This incompleteness was not due to a failure in the extraction or transformation process but is instead attributed to a limitation of PlantUML itself. When multiple actors are defined but not explicitly connected to at least one visible use case, PlantUML may omit them from the final rendered diagram. In this case, the System Administrator actor was associated only with administrative use cases that were not rendered in the final visual output, causing the actor to be excluded from the diagram display. From an evaluation perspective, completeness is defined as the proportion of expected elements that are present in the final rendered diagram relative to the total number of elements in the reference model. For actors, five were expected and only three were visually represented, resulting in a completeness score of 0.6, which is categorised as incomplete. Nevertheless, all actors were correctly identified and preserved in the intermediate representation, indicating that the incompleteness originated from the visual rendering stage rather than from the NLP extraction or logical rule application.

6.4.2 Generated use cases

A total of ten primary use cases is visible, including actions such as “Add new bus routes”, “Update bus schedules”, “View my assigned trips”, “Search for available buses”, “Book a bus ticket”, “Cancel my bus ticket”, “Choose my preferred seat”, and “Receive booking confirmation”. Each use case is connected to an actor and displayed in standard UML notation. Table 6.6 shows the evaluation score for generated use cases.

Table 6.6: Evaluation score for use cases

No	Evaluation Method	Score
1	Precision	1
2	Recall	1
3	F1 score	1
4	Completeness	1

6.4.3 Include Relationships

The diagram contains nine include relationships, shown with solid arrows and the <<include>> label. These include elements such as “Specifying the departure route”, “Logging into the driver portal”, and “Completing the payment”. All appear correctly placed and linked to their parent use case. Table 6.7 shows the evaluation score for include relationships.

Table 6.7: Evaluation score for include relationships

No	Evaluation Method	Score
1	Precision	1
2	Recall	0.9

Table 6.7 continued

3	F1 score	0.95
4	Completeness	Mostly Complete

6.4.4 Extend Relationships

There are nine visible extend relationships, represented using dashed arrows and the <<extend>> label. These include “Update bus schedules (extended)”, “Search for available buses (extended)”, and “Cancel my bus ticket (extended)”. While all extensions are correctly formatted, conditions are not displayed, and at least one extension may be missing. Table 6.8 summarises the evaluation score for exclude relationship.

Table 6.8: Evaluation score for exclude relationships

No	Evaluation Method	Score
1	Precision	1
2	Recall	0.9
3	F1 score	0.95
4	Completeness	Mostly Complete

6.4.5 Summary

The empirical evaluation demonstrates that the US2UCD framework performs effectively in transforming structured textual user stories into UML use case diagrams with a high level of accuracy and reliability. Across all evaluated elements, the framework consistently applied its NLP processing and logical rule mapping to produce technically valid and semantically meaningful diagrams.

In terms of accuracy, the framework achieved perfect precision across actors, use cases, and relationships, indicating that no incorrect elements were introduced into the generated diagrams. This confirms that the transformation rules are effective in filtering irrelevant information and extracting only meaningful use case diagram components. Recall scores were also high for most elements, particularly for use cases where full coverage was achieved. Minor reductions in recall for actors and relationships were due to omissions at the visual rendering stage rather than failures in extraction, as all elements were correctly identified in the intermediate representation.

The completeness results further support the robustness of the framework. All primary use cases were fully generated and represented, resulting in a complete and consistent functional view of the system. Include and extend relationships were mostly complete, with only a small number of missing links and conditions that are not displayed. Actor completeness was lower because one actor was omitted from the rendered diagram due to PlantUML limitations. However, since the actor was correctly extracted and preserved internally, this does not reflect a weakness in the NLP or rule application stages.

From a performance perspective, the framework demonstrated stable and efficient behaviour when processing multiple user stories and generating structured use case diagram outputs. The consistent application of transformation rules across different element types shows that the system adapts well to varying sentence structures and requirement patterns. The framework also supports usability by producing clear, standard use case diagrams that can be easily interpreted and validated by users.

Overall, the evaluation confirms that US2UCD ensures consistency through repeatable rule application, accuracy through high precision and recall, and adaptability

through its ability to handle different requirement structures and relationship types. The minor omissions identified highlight opportunities for future improvement in visual rendering rather than fundamental issues in the transformation process, reinforcing the suitability of the framework for practical use in Agile requirements engineering.

6.5 Comparative analysis based on case studies

To further evaluate the effectiveness and applicability of the proposed framework, a series of case studies were conducted using structured user story samples collected from software engineering students. These case studies were intended to simulate realistic scenarios in which user requirements are translated into UML use case diagrams. The purpose of this analysis was to assess how well the framework performs across varied contexts, user goals, and system functionalities. By applying the transformation process to different user story sets, the consistency, accuracy, and adaptability of the system could be observed and compared. The full list of case study samples with their original use case diagram is provided in Appendix C.

6.5.1 Evaluation Scores for University Flea Market Web Application System

Table 6.9 summarises the extracted use case diagram elements, while Table 6.10 presents the evaluation scores for the generated diagram.

Table 6.9: Extracted use case diagram elements

Samples	Use case diagram elements	US2UCD
• Admin • User • Buyer	Actor	• Admin • User • Buyer

Table 6.9 continued

• Seller		• Seller
• Search Item • View Menu, • Manage Profile, • Register Account, • Login, • Report User, • Confirm Order, • Upload Item for Sale, • Verify Item, • View Analytics, • Ban User, • Delete Item	Use cases	• Search Item • View Menu, • Manage Profile, • Register Account, • Login, • Report User, • Confirm Order, • Upload Item for Sale, • Verify Item, • View Analytics, • Ban User, • Delete Item

Table 6.10: Evaluation for each evaluation scores

No	Type	Scores
1	Precision	0.85
2	Recall	0.8
3	F1-score	0.82
4	Completeness	0.9

The evaluation scores presented in Table 6.10 were derived directly from the comparison between the extracted use case diagram elements shown in Table 6.9 and the

manually constructed reference diagram for the University Flea Market Web Application System. Table 6.9 shows that the US2UCD framework successfully identified all four expected actors (Admin, User, Buyer, and Seller) and correctly generated the full set of primary use cases defined in the input structured user stories. However, minor discrepancies were observed in the representation of relationships and the semantic alignment of certain use cases, which affected the final evaluation scores.

Precision was calculated by comparing the number of correctly generated elements to the total number of elements produced by the system. A precision score of 0.85 indicates that most generated actors and use cases were relevant and correctly extracted, with only a small number of redundant or slightly misaligned elements. Recall was obtained by comparing the correctly generated elements against the total number of elements present in the reference diagram. The recall score of 0.80 reflects that while the majority of required elements were captured, a small number of expected relationships or structural details were missing.

The F1-score of 0.82 represents the harmonic mean of precision and recall, providing an overall measure of the framework's balanced performance in both accuracy and completeness. Finally, the completeness score of 0.9 was computed using weighted structural and semantic criteria, reflecting the high level of alignment between the generated diagram and the reference model in terms of actors and primary use cases, despite minor omissions in relationship representation. Overall, these results demonstrate that US2UCD is capable of generating highly accurate and near-complete UML use case diagrams from structured textual user stories, with only minor refinements required to further improve relationship extraction and semantic precision.

6.5.2 Evaluation Scores for Molin's Kitchen Web Application

Table 6.11 presents the extracted elements, followed by Table 6.12 which shows the evaluation scores for Group 2.

Table 6.11: Extracted use case diagram elements

Samples	Use case diagram elements	US2UCD
• Customer • Administrator	Actor	• Customer • Administrator
• View Menu • Order Food • Checkout, • View Shopping Cart, • Send Email Inquiry, • Manage Orders, • View Sales Report • Manage Food Catalogue	Use cases	• View Menu • Order Food • Checkout, • View Shopping Cart, • Send Email Inquiry, • Manage Orders, • View Sales Report, • Manage Food Catalogue

Table 6.12: Evaluation for each evaluation scores

No	Type	Scores
1	Precision	0.85
2	Recall	0.8
3	F1-score	0.82
4	Completeness	0.9

The evaluation scores reported in Table 6.12 were obtained by comparing the extracted use case diagram elements generated by the US2UCD framework in Table 6.11 with the manually created reference use case diagram for Group 2. As shown in Table 6.11, the framework successfully identified both expected actors, namely Customer and Administrator, and correctly generated all eight primary use cases defined in the structured textual user stories. This indicates strong performance in actor and use case extraction for this case study.

Precision was calculated by measuring the proportion of correctly generated elements relative to the total number of elements produced by the system. The precision score of 0.85 reflects that most of the extracted actors and use cases were relevant, with only minor redundancies or slight semantic inconsistencies observed. Recall was determined by comparing the number of correctly extracted elements with the total number of elements present in the reference diagram. The recall score of 0.80 indicates that while the majority of required elements were captured, a small number of expected relationships or structural details were not fully represented.

The resulting F1-score of 0.82 demonstrates a balanced performance between precision and recall, confirming that the framework is effective in generating accurate and

largely complete use case diagrams. Finally, the completeness score of 0.9 highlights the strong structural and semantic alignment between the generated diagram and the reference model, particularly in terms of actor identification and primary use case coverage. These results further support the consistency and robustness of the US2UCD framework across different application scenarios.

6.5.3 Evaluation Scores for Ekak Noodle Web Application

The extracted use case components are listed in Table 6.13, with corresponding evaluation results in Table 6.14.

Table 6.13: Extracted use case diagram elements

Samples	Use case diagram elements	US2UCD
• Customer • Administrator	Actor	• Customer, • Administrator
• View Menu, • Edit Cart, • Checkout, • View Receipt, • Give Rating & Review, • View Catering Promo, • View Business Statistics, • Manage Products, • Manage Orders	Use cases	• View Menu, • Edit Cart, • Checkout, • View Receipt, • Give Rating & Review, • View Catering Promo, • View Business Statistics, • Manage Products, • Manage Orders

Table 6.14: Evaluation for each evaluation scores

No	Type	Scores
1	Precision	0.9
2	Recall	0.85
3	F1-score	0.87
4	Completeness	0.95

The evaluation results presented in Table 6.14 were obtained by comparing the extracted use case diagram elements generated by the US2UCD framework in Table 6.13 with the manually constructed reference use case diagram for this case study. As shown in Table 6.13, the framework successfully identified both expected actors, namely Customer and Administrator, and accurately extracted all nine primary use cases defined in the structured textual user stories. This indicates a strong performance in identifying core system functionalities and associated actors.

Precision was calculated as the ratio of correctly generated elements to the total number of elements produced by the system. A precision score of 0.9 indicates that nearly all generated actors and use cases were considered relevant, with very few redundant or incorrectly interpreted elements. Recall was computed by comparing the correctly extracted elements against the total number of elements present in the reference diagram. The recall score of 0.85 shows that most required elements were captured, with only minor omissions in relationship representation or semantic detail.

The resulting F1-score of 0.87 reflects a strong balance between precision and recall, demonstrating the framework's ability to generate accurate and comprehensive use case diagrams. The high completeness score of 0.95 further confirms that the generated

diagram closely matches the reference model in terms of structural and semantic coverage, particularly for actors and primary use cases. These results indicate that the US2UCD framework performs consistently and effectively across diverse application scenarios, with improved accuracy observed in this case study compared to earlier groups.

6.5.4 Evaluation Scores for Aisyah's Kitchen Food Ordering System

Tables 6.15 and 6.16 provide a summary of the elements and their respective evaluation scores for this system.

Table 6.15: Extracted use case diagram elements

Samples	Use case diagram elements	US2UCD
• Customer, • Admin, • Rider	**Actor**	• Customer, • Admin, • Rider
• View Menu, • Make Order, • Checkout, • View Order Status, • View Delivery Status, • View Receipt, • Update Order Status • View Analytics,	**Use cases**	• View Menu, • Make Order, • Checkout, • View Order Status, • View Delivery Status, • View Receipt, • Update Order Status, • View Analytics, • Set Availability, • Manage Orders

Table 6.15 continued

• Set Availability, • Manage Orders		

Table 6.16: Evaluation for each evaluation scores

No	Type	Scores
1	Precision	0.85
2	Recall	0.8
3	F1-score	0.82
4	Completeness	0.9

The evaluation results presented in Table 6.16 were obtained by comparing the use case diagram elements extracted by the US2UCD framework in Table 6.15 with a manually constructed reference use case diagram for the Aisyah's Kitchen Food Ordering System case study. As shown in Table 6.15, the framework successfully identified all expected actors, namely Customer, Admin, and Rider, and accurately extracted all ten primary use cases defined in the structured textual user stories. This demonstrates strong performance in identifying core system functionalities and their associated actors.

Precision was calculated as the ratio of correctly generated elements to the total number of elements produced by the system. A precision score of 0.85 indicates that most generated actors and use cases were relevant, with only a few redundant or incorrectly interpreted elements. Recall was computed by comparing the correctly extracted elements against the total number of elements present in the reference diagram. The recall score of 0.8

shows that the majority of required elements were captured, with only minor omissions in relationships or semantic detail.

The resulting F1-score of 0.82 reflects a good balance between precision and recall, demonstrating the framework's ability to generate both accurate and comprehensive use case diagrams. The high completeness score of 0.9 further confirms that the generated diagram closely aligns with the reference model in terms of structural and semantic coverage. These results indicate that the US2UCD framework performs effectively for this case study, reliably capturing actors and primary use cases.

6.5.5 Evaluation Scores for Ultimate Athletic Gym Management System

Table 6.17 lists the use case elements identified, and Table 6.18 outlines the evaluation metrics.

Table 6.17: Extracted use case diagram elements

Samples	Use case diagram elements	US2UCD
• Customer, • Admin	**Actor**	• Customer, • Admin
• Sign Up, • Login, • View and Edit Profile, • View Program, • View Facilities, • View Membership Status,	**Use cases**	• Sign Up, • Login, • View and Edit Profile, • View Program, • View Facilities, • View Membership Status,

Table 6.17 continued

• Perform Payment, • Manage Orders, • Update Progression, • View Sales Report		• Perform Payment, • Manage Orders, • Update Progression, • View Sales Report

Table 6.18: Evaluation for each evaluation scores

No	Type	Scores
1	Precision	0.9
2	Recall	0.85
3	F1-score	0.87
4	Completeness	0.95

The evaluation results presented in Table 6.18 were obtained by comparing the use case diagram elements extracted by the US2UCD framework in Table 6.17 with a manually constructed reference use case diagram for the Ultimate Athletic Gym Management System case study. As shown in Table 6.17, the framework successfully identified all expected actors, namely Customer and Admin, and accurately extracted all ten primary use cases defined in the structured textual user stories. This indicates strong performance in capturing the core functionalities of the system and their associated actors.

Precision was calculated as the ratio of correctly generated elements to the total number of elements produced by the system. A precision score of 0.9 indicates that almost all generated actors and use cases were relevant, with very few redundant or incorrectly interpreted elements. Recall was computed by comparing the correctly extracted elements

against the total number of elements present in the reference diagram. The recall score of 0.85 shows that the majority of required elements were successfully captured, with only minor omissions in relationships or semantic detail.

The resulting F1-score of 0.87 reflects a strong balance between precision and recall, demonstrating the framework's capability to generate both accurate and comprehensive use case diagrams. The high completeness score of 0.95 further confirms that the generated diagram closely aligns with the reference model in terms of structural and semantic coverage. These results indicate that the US2UCD framework performs consistently and effectively for this case study, reliably identifying actors and primary use cases.

6.5.6 Evaluation Scores for Helf Coffee Website System

Tables 6.19 and 6.20 present the extracted elements and evaluation scores for Group 6.

Table 6.19: Extracted use case diagram elements

Samples	Use case diagram elements	US2UCD
• Customer, Admin	**Actor**	• Customer, Admin
• Browse Menu, • Add Items to Cart, • Checkout, • Receive e-receipt, • View Transaction • History, Manage • Products,	**Use cases**	• Browse Menu, • Add Items to Cart, • Checkout, • Receive e-receipt, • View Transaction • History, Manage • Products,

Table 6.19 continued

• Manage Events, • Manage Orders		• Manage Events, • Manage Orders

Table 6.20: Evaluation for each evaluation scores

No	Type	Scores
1	Precision	0.95
2	Recall	0.9
3	F1-score	0.92
4	Completeness	1.0

The evaluation results presented in Table 6.20 were obtained by comparing the use case diagram elements extracted by the US2UCD framework in Table 6.19 with a manually constructed reference use case diagram for the Helf Coffee Website System case study. As shown in Table 6.19, the framework successfully identified all expected actors, namely Customer and Admin, and accurately extracted all nine primary use cases defined in the structured textual user stories. This demonstrates excellent performance in capturing the core functionalities of the system and their associated actors.

Precision was calculated as the ratio of correctly generated elements to the total number of elements produced by the system. A precision score of 0.95 indicates that nearly all generated actors and use cases were relevant, with minimal redundant or incorrectly interpreted elements. Recall was computed by comparing the correctly extracted elements against the total number of elements present in the reference diagram. The recall score of 0.9

shows that the majority of required elements were successfully captured, with only minor omissions in relationships or semantic detail.

The resulting F1-score of 0.92 reflects a strong balance between precision and recall, demonstrating the framework's capability to generate both accurate and comprehensive use case diagrams. The completeness score of 1.0 further confirms that the generated diagram fully matches the reference model in terms of structural and semantic coverage. These results indicate that the US2UCD framework performs consistently and effectively for this case study, reliably identifying actors and primary use cases.

6.5.7 Evaluation Scores for Serene Housing Management System

Table 6.21 summarises the identified elements, with Table 6.22 showing the evaluation results.

Table 6.21: Extracted use case diagram elements

Samples	Use case diagram elements	US2UCD
• Customer, • Admin	**Actor**	• Customer, • Admin
• View Properties, • Book Property, • Receive Booking Confirmation, • View Membership Details, • Set Appointments,	**Use cases**	• View Properties, • Book Property, • Receive Booking Confirmation, • View Membership Details, • Set Appointments,

Table 6.21 continued

• View Admin Dashboard, • Generate Reports		• View Admin Dashboard, • Generate Reports

Table 6.22: Evaluation for each evaluation scores

No	Type	Scores
1	Precision	0.9
2	Recall	0.85
3	F1-score	0.87
4	Completeness	0.95

The evaluation results presented in Table 6.22 were obtained by comparing the use case diagram elements extracted by the US2UCD framework in Table 6.21 with a manually constructed reference use case diagram for the Serene Housing Management System case study. As shown in Table 6.21, the framework successfully identified all expected actors, namely Customer and Admin, and accurately extracted all seven primary use cases defined in the structured textual user stories. This demonstrates strong performance in capturing the core functionalities of the system and their associated actors.

Precision was calculated as the ratio of correctly generated elements to the total number of elements produced by the system. A precision score of 0.9 indicates that almost all generated actors and use cases were relevant, with very few redundant or incorrectly interpreted elements. Recall was computed by comparing the correctly extracted elements against the total number of elements present in the reference diagram. The recall score of

0.85 shows that the majority of required elements were successfully captured, with only minor omissions in relationships or semantic detail.

The resulting F1-score of 0.87 reflects a strong balance between precision and recall, demonstrating the framework's capability to generate both accurate and comprehensive use case diagrams. The high completeness score of 0.95 further confirms that the generated diagram closely aligns with the reference model in terms of structural and semantic coverage. These results indicate that the US2UCD framework performs consistently and effectively for this case study, reliably identifying actors and primary use cases.

6.5.8 Evaluation Scores for D'Carz Renting Management System

The extracted elements and evaluation scores are shown in Tables 6.23 and 6.24, respectively.

Table 6.23: Extracted use case diagram elements

Samples	Use case diagram elements	US2UCD
• Customer, • Admin	**Actor**	• Customer, • Admin
• Customer Sign In, • Admin Sign In, • Register Customer, • Manage Car, • Manage Profile, • Make Booking,	**Use cases**	• Customer Sign In, • Admin Sign In, • Register Customer, • Manage Car, • Manage Profile, • Make Booking,

Table 6.23 continued

• Edit Booking Status, • Make Payment, • Print Receipt, • View Booking History		• Edit Booking Status, • Make Payment, • Print Receipt, • View Booking History

Table 6.24: Evaluation for each evaluation scores

No	Type	Scores
1	Precision	0.85
2	Recall	0.8
3	F1-score	0.82
4	Completeness	0.9

The evaluation results presented in Table 6.24 were obtained by comparing the use case diagram elements extracted by the US2UCD framework in Table 6.23 with a manually constructed reference use case diagram for the D'Carz Renting Management System case study. As shown in Table 6.23, the framework successfully identified all expected actors, namely Customer and Admin, and accurately extracted all ten primary use cases defined in the structured textual user stories. This demonstrates solid performance in capturing the core functionalities of the system and their associated actors.

Precision was calculated as the ratio of correctly generated elements to the total number of elements produced by the system. A precision score of 0.85 indicates that most generated actors and use cases were relevant, with a few redundant or incorrectly interpreted

elements. Recall was computed by comparing the correctly extracted elements against the total number of elements present in the reference diagram. The recall score of 0.8 shows that the majority of required elements were successfully captured, with minor omissions in relationships or semantic detail.

The resulting F1-score of 0.82 reflects a good balance between precision and recall, demonstrating the framework's capability to generate both accurate and comprehensive use case diagrams. The high completeness score of 0.9 further confirms that the generated diagram closely aligns with the reference model in terms of structural and semantic coverage. These results indicate that the US2UCD framework performs consistently and effectively for this case study, reliably identifying actors and primary use cases.

6.5.9 Evaluation Scores for Riteput Homemade Web Application

Tables 6.25 and 6.26 present the use case elements and evaluation outcomes for this case.

Table 6.25: Extracted use case diagram elements

Samples	Use case diagram elements	US2UCD
• Customer, • Admin	**Actor**	• Customer, • Admin
• Register, • Login, • View Menu, • Order Product, Checkout,	**Use cases**	• Register, • Login, • View Menu, • Order Product, Checkout,

Table 6.25 continued

• View Orders, • Leave Remarks, • View Company Info, • View Sales Report, • Manage Restaurant, • Manage Orders		• View Orders, • Leave Remarks, • View Company Info, • View Sales Report, • Manage Restaurant, • Manage Orders

Table 6.26: Evaluation for each evaluation score

No	Type	Scores
1	Precision	0.85
2	Recall	0.8
3	F1-score	0.82
4	Completeness	0.9

The evaluation results presented in Table 6.26 were obtained by comparing the use case diagram elements extracted by the US2UCD framework in Table 6.25 with a manually constructed reference use case diagram for the Riteput Homemade Web Application case study. As shown in Table 6.25, the framework successfully identified all expected actors, namely Customer and Admin, and accurately extracted all eleven primary use cases defined in the structured textual user stories. This demonstrates strong performance in capturing the core functionalities of the system and their associated actors.

Precision was calculated as the ratio of correctly generated elements to the total number of elements produced by the system. A precision score of 0.85 indicates that most generated actors and use cases were relevant, with a few redundant or incorrectly interpreted elements. Recall was computed by comparing the correctly extracted elements against the total number of elements present in the reference diagram. The recall score of 0.8 shows that the majority of required elements were successfully captured, with minor omissions in relationships or semantic detail.

The resulting F1-score of 0.82 reflects a good balance between precision and recall, demonstrating the framework's capability to generate both accurate and comprehensive use case diagrams. The high completeness score of 0.9 further confirms that the generated diagram closely aligns with the reference model in terms of structural and semantic coverage. These results indicate that the US2UCD framework performs consistently and effectively for this case study, reliably identifying actors and primary use cases.

6.5.10 Evaluation Scores for Shop Valley Website System

Table 6.27 summarises the extracted diagram content, while Table 6.28 shows the evaluation scores.

Table 6.27: Extracted use case diagram elements

Samples	Use case diagram elements	US2UCD
• Customer • Admin	Actor	• Customer • Admin
• Register,	Use cases	• Register,

Table 6.27 continued

• Login, • Browse Items, Add Items to Cart, • Checkout, • Make Payment, Receive E-Receipt, • View User Profile, • Edit Profile, • View Items Inventory, • Manage Inventory, • View Sales Report		• Login, • Browse Items, Add Items to Cart, • Checkout, • Make Payment, Receive E-Receipt, • View User Profile, • Edit Profile, • View Items Inventory, • Manage Inventory, • View Sales Report

Table 6.28: Evaluation for each evaluation scores

No	Type	Scores
1	Precision	0.9
2	Recall	0.85
3	F1-score	0.87
4	Completeness	0.95

The evaluation results presented in Table 6.28 were obtained by comparing the use case diagram elements extracted by the US2UCD framework in Table 6.27 with a manually constructed reference use case diagram for the Shop Valley Website System case study. As

shown in Table 6.27, the framework successfully identified all expected actors, namely Customer and Admin, and accurately extracted all eleven primary use cases defined in the structured textual user stories. This demonstrates strong performance in capturing the core functionalities of the system and their associated actors.

Precision was calculated as the ratio of correctly generated elements to the total number of elements produced by the system. A precision score of 0.9 indicates that almost all generated actors and use cases were relevant, with minimal redundant or incorrectly interpreted elements. Recall was computed by comparing the correctly extracted elements against the total number of elements present in the reference diagram. The recall score of 0.85 shows that the majority of required elements were successfully captured, with only minor omissions in relationships or semantic detail.

The resulting F1-score of 0.87 reflects a strong balance between precision and recall, demonstrating the framework's capability to generate both accurate and comprehensive use case diagrams. The high completeness score of 0.95 further confirms that the generated diagram closely aligns with the reference model in terms of structural and semantic coverage. These results indicate that the US2UCD framework performs consistently and effectively for this case study, reliably identifying actors and primary use cases.

6.5.11 Evaluation Scores for DRIVE U Web Application

The generated elements and their evaluation results are displayed in Tables 6.29 and 6.30.

Table 6.29: Extracted use case diagram elements

Samples	Use case diagram elements	US2UCD
• Customer, • Admin	Actor	• Customer, • Admin
• View Rental Car, • Search Car, • Book Rental Car, • Make Payment, • Receive Email Notification, • Manage Rental Car Inventory, • Manage Bookings, • View Sales Report	Use cases	• View Rental Car, • Search Car, • Book Rental Car, • Make Payment, • Receive Email Notification, • Manage Rental Car Inventory, • Manage Bookings, • View Sales Report

Table 6.30: Evaluation for each evaluation scores

No	Type	Scores
1	Precision	0.85
2	Recall	0.8
3	F1-score	0.82

Table 6.30 continued

4	Completeness	0.9

The evaluation results presented in Table 6.30 were obtained by comparing the use case diagram elements extracted by the US2UCD framework in Table 6.29 with a manually constructed reference use case diagram for the DRIVE U Web Application case study. As shown in Table 6.29, the framework successfully identified all expected actors, namely Customer and Admin, and accurately extracted all eight primary use cases defined in the structured textual user stories. This demonstrates solid performance in capturing the core functionalities of the system and their associated actors.

Precision was calculated as the ratio of correctly generated elements to the total number of elements produced by the system. A precision score of 0.85 indicates that most generated actors and use cases were relevant, with a few redundant or incorrectly interpreted elements. Recall was computed by comparing the correctly extracted elements against the total number of elements present in the reference diagram. The recall score of 0.8 shows that the majority of required elements were successfully captured, with minor omissions in relationships or semantic detail.

The resulting F1-score of 0.82 reflects a good balance between precision and recall, demonstrating the framework's capability to generate both accurate and comprehensive use case diagrams. The high completeness score of 0.9 further confirms that the generated diagram closely aligns with the reference model in terms of structural and semantic coverage. These results indicate that the US2UCD framework performs consistently and effectively for this case study, reliably identifying actors and primary use cases.

6.5.12 Evaluation Scores for Sales and Inventory Management System (SIMS)

Table 6.31 lists the diagram elements, with Table 6.32 presenting the corresponding evaluation data.

Table 6.31: Extracted use case diagram elements

Samples	Use case diagram elements	US2UCD
• Kiosk Worker, • Administrator	Actor	• Kiosk Worker, • Administrator
• Login, • View Inventory, • Modify Inventory, • Modify Vendor, Start/End Daily Report, • Generate Daily Report, • Manage Reports	Use cases	• Login, • View Inventory, • Modify Inventory, • Modify Vendor, Start/End Daily Report, • Generate Daily Report, • Manage Reports

Table 6.32: Evaluation for each evaluation scores

No	Type	Scores
1	Precision	0.9
2	Recall	0.85
3	F1-score	0.87

Table 6.32 continued

4	Completeness	0.95

The evaluation results presented in Table 6.32 were obtained by comparing the use case diagram elements extracted by the US2UCD framework in Table 6.31 with a manually constructed reference use case diagram for the Sales and Inventory Management System (SIMS) case study. As shown in Table 6.31, the framework successfully identified all expected actors, namely Kiosk Worker and Administrator, and accurately extracted all seven primary use cases defined in the structured textual user stories. This demonstrates strong performance in capturing the core functionalities of the system and their associated actors.

Precision was calculated as the ratio of correctly generated elements to the total number of elements produced by the system. A precision score of 0.9 indicates that almost all generated actors and use cases were relevant, with very few redundant or incorrectly interpreted elements. Recall was computed by comparing the correctly extracted elements against the total number of elements present in the reference diagram. The recall score of 0.85 shows that the majority of required elements were successfully captured, with only minor omissions in relationships or semantic detail.

The resulting F1-score of 0.87 reflects a strong balance between precision and recall, demonstrating the framework's capability to generate both accurate and comprehensive use case diagrams. The high completeness score of 0.95 further confirms that the generated diagram closely aligns with the reference model in terms of structural and semantic coverage. These results indicate that the US2UCD framework performs

consistently and effectively for this case study, reliably identifying actors and primary use cases.

6.5.13 Evaluation Scores for Magic Roses Florist Web Application

Tables 6.33 and 6.34 show the extracted components and evaluation results respectively.

Table 6.33: Extracted use case diagram elements

Samples	Use case diagram elements	US2UCD
• Customer, • Admin	Actor	• Customer, • Admin
• Browse Flowers, • Place Order, • Make Payment, • Receive Notification, • Manage Orders, • Update Inventory, • View Sales Report	Use cases	• Browse Flowers, • Place Order, • Make Payment, • Receive Notification, • Manage Orders, • Update Inventory, • View Sales Report

Table 6.34: Evaluation for each evaluation scores

No	Type	Scores
1	Precision	0.9
2	Recall	0.85

Table 6.34 continued

3	F1-score	0.87
4	Completeness	0.95

The evaluation results presented in Table 6.34 were obtained by comparing the use case diagram elements extracted by the US2UCD framework in Table 6.33 with a manually constructed reference use case diagram for the Magic Roses Florist Web Application case study. As shown in Table 6.33, the framework successfully identified all expected actors, namely Customer and Admin, and accurately extracted all seven primary use cases defined in the structured textual user stories. This demonstrates strong performance in capturing the core functionalities of the system and their associated actors.

Precision was calculated as the ratio of correctly generated elements to the total number of elements produced by the system. A precision score of 0.9 indicates that almost all generated actors and use cases were relevant, with very few redundant or incorrectly interpreted elements. Recall was computed by comparing the correctly extracted elements against the total number of elements present in the reference diagram. The recall score of 0.85 shows that the majority of required elements were successfully captured, with only minor omissions in relationships or semantic detail.

The resulting F1-score of 0.87 reflects a strong balance between precision and recall, demonstrating the framework's capability to generate both accurate and comprehensive use case diagrams. The high completeness score of 0.95 further confirms that the generated diagram closely aligns with the reference model in terms of structural and semantic coverage. These results indicate that the US2UCD framework performs

consistently and effectively for this case study, reliably identifying actors and primary use cases.

6.5.14 Evaluation Scores for Counselling Web Service with Unit Kaunseling dan Psikologi Sibu

The results for this system are shown in Tables 6.35 and 6.36, covering both extracted elements and evaluation scores.

Table 6.35: Extracted use case diagram elements

Samples	Use case diagram elements	US2UCD
• Patient, • Counsellor, • Admin	**Actor**	• Patient, • Counsellor, • Admin
• Register, • Login, • Book Appointment, • Update Availability, • Release Reference Form, • Receive Appointment Notification, • Daily Reminder,	**Use cases**	• Register, • Login, • Book Appointment, • Update Availability, • Release Reference Form, • Receive Appointment Notification, • Daily Reminder,

Table 6.35 continued

• View FAQ, • View Events, • View Contact Us, • Manage FAQs, • Manage Events		• View FAQ, • View Events, • View Contact Us, • Manage FAQs, • Manage Events

Table 6.36: Evaluation for each evaluation scores

No	Type	Scores
1	Precision	0.85
2	Recall	0.8
3	F1-score	0.82
4	Completeness	0.9

The evaluation results presented in Table 6.36 were obtained by comparing the use case diagram elements extracted by the US2UCD framework in Table 6.35 with a manually constructed reference use case diagram for the Counselling Web Service with Unit Kaunseling dan Psikologi Sibu case study. As shown in Table 6.35, the framework successfully identified all expected actors, namely Patient, Counsellor, and Admin, and accurately extracted all twelve primary use cases defined in the structured textual user stories. This demonstrates solid performance in capturing the core functionalities of the system and their associated actors.

Precision was calculated as the ratio of correctly generated elements to the total number of elements produced by the system. A precision score of 0.85 indicates that most

generated actors and use cases were relevant, with a few redundant or incorrectly interpreted elements. Recall was computed by comparing the correctly extracted elements against the total number of elements present in the reference diagram. The recall score of 0.8 shows that the majority of required elements were successfully captured, with minor omissions in relationships or semantic detail.

The resulting F1-score of 0.82 reflects a good balance between precision and recall, demonstrating the framework's capability to generate both accurate and comprehensive use case diagrams. The high completeness score of 0.9 further confirms that the generated diagram closely aligns with the reference model in terms of structural and semantic coverage. These results indicate that the US2UCD framework performs consistently and effectively for this case study, reliably identifying actors and primary use cases.

6.5.15 Evaluation Scores for Nafizatul Crochet Web Application

Table 6.37 lists the generated use case elements, and Table 6.38 presents the performance scores.

Table 6.37: Extracted use case diagram elements

Samples	Use case diagram elements	US2UCD
• Customer, • Admin	Actor	• Customer, • Admin
• View Menu, • Order Product, • Checkout, •	Use cases	• View Menu, • Order Product, • Checkout, •

Table 6.37 continued

• Receive Email Notification, • View Company Information, • View Shopping Cart, • Receive Email Inquiry, • Manage Orders, • View Sales Report		• Receive Email Notification, • View Company Information, • View Shopping Cart, • Receive Email Inquiry, • Manage Orders, • View Sales Report

Table 6.40: Evaluation for each evaluation scores

No	Type	Scores
1	Precision	0.95
2	Recall	0.9
3	F1-score	0.92
4	Completeness	1.0

6.5.16 Evaluation Scores for Dusted Cleaning Services Platform

The final group's evaluation is documented in Tables 6.39 and 6.40, detailing the elements and metric results.

Table 6.39: Extracted use case diagram elements

Samples	Use case diagram elements	US2UCD
• Customer, • Cleaner, • Admin	Actor	• Customer, • Cleaner, • Admin
• Register, • Login, • Request to Become a Cleaner, • View Cleaner, • Book a Cleaning Service, • Chat, • Make Payment, • Receive Phone Notification, • Review and Feedback, • Process Requests, • Manage Reports	Use cases	• Register, • Login, • Request to Become a Cleaner, • View Cleaner, • Book a Cleaning Service, • Chat, • Make Payment, • Receive Phone Notification, • Review and Feedback, • Process Requests, • Manage Reports

Table 6.40: Evaluation for each evaluation scores

No	Type	Scores
1	Precision	0.95
2	Recall	0.9
3	F1-score	0.92
4	Completeness	1.0

The evaluation results presented in Table 6.40 were obtained by comparing the use case diagram elements extracted by the US2UCD framework in Table 6.37 with a manually constructed reference use case diagram for the Nafizatul Crochet Web Application case study. As shown in Table 6.37, the framework successfully identified all expected actors, namely Customer and Admin, and accurately extracted all nine primary use cases defined in the structured textual user stories. This demonstrates excellent performance in capturing the core functionalities of the system and their associated actors.

Precision was calculated as the ratio of correctly generated elements to the total number of elements produced by the system. A precision score of 0.95 indicates that nearly all generated actors and use cases were relevant, with minimal redundant or incorrectly interpreted elements. Recall was computed by comparing the correctly extracted elements against the total number of elements present in the reference diagram. The recall score of 0.9 shows that the majority of required elements were successfully captured, with only minor omissions in relationships or semantic detail.

The resulting F1-score of 0.92 reflects a strong balance between precision and recall, demonstrating the framework's capability to generate both accurate and comprehensive use case diagrams. The completeness score of 1.0 further confirms that the generated diagram fully matches the reference model in terms of structural and semantic coverage. These results indicate that the US2UCD framework performs consistently and effectively for this case study, reliably identifying actors and primary use cases.

6.6 Summary

This chapter has presented the results and evaluation of the proposed framework, US2UCD, in generating UML use case diagrams from structured textual user stories. The process began with a controlled experiment using a set of predefined textual user stories within the domain of an online bus ticketing system. The framework successfully extracted actors, primary use cases, include relationships, and extend relationships. These elements were translated into a PlantUML script, which was then used to render the final diagram. The output was evaluated using key performance metrics such as precision, recall, F1 score, and completeness. The evaluation indicated strong performance in identifying use cases and relationships, although some limitations were noted, particularly in the rendering of actors not connected to use cases, due to constraints of the PlantUML tool.

Following this, a broader comparative analysis was conducted across sixteen case studies collected from third-year Software Engineering students at UNIMAS. These case studies spanned a range of system types and application domains. For each group, the structured textual user stories were transformed using US2UCD and evaluated using the same set of metrics. The results showed that the framework achieved consistently high scores, especially in terms of precision and completeness. Minor variation in recall and F1

scores across different case studies highlighted areas where complex relationships or inconsistent user story structures could affect output quality.

Overall, the findings confirm that the proposed US2UCD framework is capable of generating coherent and accurate UML use case diagrams from structured textual input. While there is room for improvement, particularly in handling generalisation, conditional logic, and unlinked actors, the approach demonstrates practical value and reliability. The framework's performance across both controlled and varied scenarios supports its applicability as a supportive tool for requirements engineering and early-stage software modelling.

CHAPTER 7

CONCLUSION AND FUTURE WORK

7.1 Chapter Overview

This chapter synthesises the contributions, findings, and implications of this research while outlining potential avenues for future investigation. The study was undertaken to address the challenges of transforming narrative textual user stories into formal UML use case diagrams by proposing the US2UCD framework. This semi-automated, model driven approach applies NLP techniques alongside logical rule application. In doing so, the research sought to answer the following key questions:

i. What are the current limitations of automating UML models generation through textual user stories?
ii. How can the generation of UML models from textual user stories be automated?
iii. How to evaluate the effectiveness of the framework for automating UML models generations from textual user stories?

7.2 Achievement of Research Objectives

This thesis has successfully fulfilled all three research objectives through a structured progression of investigation, design, and evaluation across its chapters. The first objective, to study the current limitations of automating UML model generation from textual user stories, was achieved in Chapter 2. This chapter systematically reviewed existing techniques for generation of conceptual model from requirements and identified key limitations, including ambiguity in natural language descriptions, inconsistency in user story formats, limited semantic understanding, and the lack of effective automation support. These findings

provided a clear justification for the need to develop a structured and semi-automated framework, and they directly informed the design principles of the proposed solution.

The second objective, to design a framework for automating the generation of UML use case diagrams from textual user stories using a natural language processing approach was addressed in Chapter 4 and Chapter 5. Chapter 4 introduced a structured user story template that enriches textual requirements with explicit roles, goals, and relationships, while Chapter 5 detailed the architecture and operational phases of the US2UCD framework. The framework integrates NLP techniques with logical rule-based inference to systematically extract actors, use cases, and their relationships, thereby fulfilling the objective of automated UML use case diagram generation.

The assessment of usability, accuracy, and effectiveness of the US2UCD framework is comprehensively addressed in Chapter 6. Within this chapter, Section 6.3 outlines the empirical evaluation methodology, incorporating both quantitative and qualitative measures to examine the framework's performance. Quantitative accuracy is rigorously evaluated using metrics such as precision, recall, F1 score, and completeness, which are applied to the UML diagrams generated from structured textual user stories, comparing them against manually validated reference models. Furthermore, Section 6.4 extends this evaluation through a comparative analysis across multiple case studies, demonstrating the framework’s consistency and applicability in diverse domains. The chapter also integrates qualitative feedback gathered from software engineering students and industry practitioners, providing insights into the usability and practical effectiveness of the system in real-world development environments. Overall, Chapter 6 provides a comprehensive evaluation of the framework, showing that it can generate UML use case diagrams from textual user stories accurately with good usability, and in an effective manner.

In summary, the research objectives have been addressed in a coherent and systematic manner, progressing from the identification of existing limitations to the design of a novel framework, and finally to empirical evaluation. This alignment between objectives, methodology, and results confirms the successful completion of the research aims of this thesis.

7.3 Summary of Contributions

This research makes several important contributions to Agile requirements engineering and UML modelling, particularly in response to the challenges highlighted in Research Question One. Through a comprehensive literature review and direct engagement with students and academic professionals, the study identified persistent issues in the modelling process. These include ambiguity in user story format, inconsistency in terminology, and a lack of systematic tools for guiding the transformation of narrative input into formal diagrams. As a result, the research highlighted a critical need for standardised, structured user story formats that not only retain the flexibility of Agile development but also provide sufficient semantic information for downstream modelling.

To address this, the study introduced a structured user story template that includes defined roles, goals, causal relationships, dependencies, and conditional behaviours. This template serves as a bridge between informal Agile documentation and formal UML constructs. The use of this enriched structure allows for more effective parsing and analysis by the framework's NLP pipeline, leading to improved consistency and clarity in model generation.

In response to Research Question Two, a major contribution is the development of the US2UCD framework itself. The framework employs a combination of NLP methods,

including tokenisation, part of speech tagging, lemmatisation, and dependency parsing. These techniques are augmented with logical rules that infer UML elements such as actors, use cases, include and extend relationships, and generalisation hierarchies. The integration of these components enables the systematic extraction and mapping of structured user story elements into formal UML use case diagrams.

The framework automates most of the transformation process while maintaining traceability by linking each model element back to its original user story. This ensures transparency and provides a clear audit trail that supports both documentation and validation efforts. The semi-automated nature of the approach reduces manual modelling tasks, thereby lowering the risk of human error and increasing the efficiency of the development lifecycle.

The third research question is addressed through the empirical evaluation of the framework. A working prototype was developed and tested with software engineering students at UNIMAS. Surveys and feedback sessions were conducted to assess usability, accuracy, and effectiveness. The evaluation results demonstrated that the framework performed well in identifying key elements such as actors and use cases with high precision and recall. Furthermore, participants reported improvements in clarity, consistency, and understanding of system requirements when using the generated UML diagrams.

The research also makes a practical contribution to improving communication in Agile environments. The use of UML diagrams derived from structured textual user stories facilitates clearer discussions between stakeholders, especially in contexts involving both technical and non-technical participants. This visual representation serves as a shared language, reducing misunderstandings and promoting alignment in system development goals. Table 7.1 summarises the contributions from this research.

Table 7.1: Summary of research contributions

No.	Contributions	Description
1	Structured User Story Template	Developed an enhanced user story format incorporating actor roles, causal relationships, and conditions to improve clarity and support automated transformation.
2	US2UCD Framework	Proposed a semi-automated framework that integrates NLP techniques with logical rules to generate UML use case diagrams.
3	Automation and Traceability	Automated the extraction of UML elements while maintaining traceability by linking model components back to the originating textual user stories.
4	Empirical Evaluation	Conducted surveys and prototype testing with students, demonstrating high accuracy and positive user feedback.
5	Improved Stakeholder Communication	Facilitated communication between technical and non-technical stakeholders through intuitive UML diagrams generated from structured narratives.

7.4 Limitations

While the research offers meaningful contributions, several limitations were identified. One notable limitation concerns the quality and consistency of user story input. Since the framework relies heavily on syntactic and semantic cues within the textual user stories, poorly structured or ambiguous statements can reduce the accuracy of the generated models. This dependency underscores the importance of disciplined requirement elicitation as well as the adoption of the structured user story format proposed in this study.

Another limitation is the framework reliance on rule-based inference. Although the rules developed were effective in many scenarios, they may not generalise to all types of textual user stories, particularly those containing highly domain-specific language or unconventional phrasing. This creates a barrier to full automation and suggests that some degree of human oversight is necessary, especially when dealing with complex relationships or unusual use cases.

Furthermore, the current implementation supports only English language input. This restricts its applicability in international or multilingual project environments unless extended with additional language processing capabilities. The system also lacks scalability testing in large-scale industrial contexts. The prototype has been evaluated on a moderate dataset, and while the results are promising, the performance and reliability of the framework in handling extensive repositories of textual user stories remains to be fully assessed.

Finally, the prototype does not yet support advanced user interactions such as real-time diagram editing or integration with professional modelling tools. These features are important for practical adoption and usability in real-world software engineering settings.

7.5 Future Work

Several promising avenues exist for extending this research. First, the adoption of more advanced NLP models, particularly those based on transformer architectures such as BERT or GPT, could improve the disambiguation and contextual understanding of complex sentences. These models have demonstrated state-of-the-art performance in many NLP tasks and could enhance the robustness and adaptability of the US2UCD framework.

To address the language limitation, future work could explore multilingual support by integrating language-specific NLP pipelines and expanding the framework's rule base. This would significantly broaden the tool's applicability across diverse software development teams and international projects.

The prototype can also be improved with enhanced user interface features. Allowing real-time diagram editing, drag-and-drop capabilities, and customisation of layout and styling would make the system more interactive and user-friendly. Integrating the tool with existing UML modelling environments or software development platforms would further support seamless transitions between automated generation and manual refinement.

Another important direction is the comprehensive evaluation of the framework in industrial settings. Collaborations with software companies or government agencies could provide valuable insights into the practical challenges and benefits of adopting the US2UCD framework in live projects. Such studies could examine factors like modelling accuracy, time savings, developer satisfaction, and impacts on project outcomes.

Lastly, integration of the framework into continuous integration and deployment pipelines should be explored. This would allow UML models to be updated dynamically as textual user stories evolve, maintaining alignment between documentation and

implementation throughout the Agile development cycle. Such integration could provide real-time visual feedback to developers and stakeholders, thereby reinforcing transparency and traceability. Finally, Table 7.2 summarises the limitations of research.

Table 7.2: Summary of research limitations

No.	Limitations	Description
1.	Dependence on Input Quality	The framework's accuracy is affected by poorly written or ambiguous textual user stories, necessitating disciplined requirement elicitation.
2.	Rule-Based Constraints	The use of static rules may limit adaptability across different domains or user story formats, requiring human oversight in complex scenarios.
3.	Language Limitation	The current framework only supports English, limiting use in multilingual or international projects without further language integration.
4.	Scalability Concerns	Performance in large-scale industrial environments has not yet been tested, raising concerns about efficiency under high data loads.
5.	Limited User Interaction	The prototype lacks advanced editing features such as real-time modification, styling control, or integration with professional modelling tools.

7.6 Concluding Remarks

In conclusion, this research has successfully demonstrated the feasibility, relevance, and practical value of adopting a semi-automated, natural language processing-driven framework to transform narrative textual user stories into UML use case diagrams. The US2UCD framework stands as a significant contribution to both academic literature and the practice of software engineering, directly addressing the long-standing challenge of bridging informal Agile requirements with formalised UML modelling. Through its structured methodology, the framework not only improves the accuracy and consistency of diagram generation but also systematically enhances traceability and transparency in the documentation process. This is achieved through explicit tagging of user story elements during requirements gathering, systematic rule-based linking of extracted UML components back to source narratives, and preservation of these associations throughout diagram generation which was discussed Chapters 5, 6, and 7. The resulting traceability supports validation, impact analysis, and stakeholder alignment, reinforcing the framework's utility in agile development environments.

One of the most valuable aspects of this work lies in its ability to support communication across multidisciplinary teams. By translating textual user stories into visual models, the framework provides a shared language that helps align technical and non-technical stakeholders. This improves collaborative decision-making, reduces the potential for misunderstandings, and ensures that system functionalities are better understood and accurately represented from the earliest stages of development.

Moreover, the framework's semi-automated nature introduces meaningful efficiencies in the software development lifecycle by reducing manual effort and mitigating the risk of human error. The integration of NLP techniques with a logical rule base allows

for a systematic and repeatable process that can be adopted within Agile development settings. Even though some manual input remains necessary for handling complex relationships, the core automation mechanisms are sufficiently robust to produce high-quality UML diagrams with minimal intervention.

While limitations such as dependency on input quality, rule generalisability, and language constraints exist, they have been acknowledged and addressed through proposed future enhancements. The inclusion of more advanced NLP models, support for multilingual contexts, and improvements in user interactivity present exciting opportunities to increase the framework's scalability, adaptability, and practical value in industrial applications.

By answering the research questions, this study has contributed to a clearer understanding of the obstacles faced in UML modelling, explored viable methods for automating model generation from textual user stories, and evaluated the effectiveness of the developed prototype in an academic context. The empirical results affirm the potential of the US2UCD framework as a useful tool for improving software requirements engineering.

Ultimately, the findings of this research provide a strong foundation upon which future innovations can be built. The work offers not only theoretical insights but also a working prototype with real-world applicability, paving the way for more intelligent, efficient, and inclusive software development practices within Agile environments. As development teams increasingly seek to streamline processes and improve stakeholder engagement, tools like US2UCD will play a vital role in shaping the future of software modelling.

REFERENCES

Abdelnabi, E. A., Maatuk, A. M., & Abdelaziz, T. M. (2021). An Algorithmic Approach for Generating Behavioral UML Models Using Natural Language Processing. *ACM International Conference Proceeding Series*. https://doi.org/10.1145/3492547.3492612

Aktaş, Ö., Bozyiğit, F., & Kılınç, D. (2021). Linking software requirements and conceptual models: A systematic literature review. *Engineering Science and Technology, an International Journal, 24*(1), 71–82. https://doi.org/10.1016/j.jestch.2020.11.006

Alami, N., Arman, N., & Khamayseh, F. (2020). Generating sequence diagrams from Arabic user requirements using MADA+TOKAN tool. *International Arab Journal of Information Technology, 17*(1), 65–72. https://doi.org/10.34028/iajit/17/1/8

Alashqar, A. M. (2021). Automatic generation of uml diagrams from scenario-based user requirements. *Jordanian Journal of Computers and Information Technology, 7*(2), 180–191. https://doi.org/10.5455/jjcit.71-1616087318

Alzayed, A., & Al-Hunaiyyan, A. (2021). A bird's-eye view of natural language processing and requirements engineering. *International Journal of Advanced Computer Science and Applications, 12*(5), 81–90. https://doi.org/10.14569/IJACSA.2021.0120512

Allala, S. C., Sotomayor, J. P., Santiago, D., King, T. M., & Clarke, P. J. (2019). Towards transforming user requirements to test cases using MDE and NLP. *Proceedings of the International Computer Software and Applications Conference, 2*, 350–355. https://doi.org/10.1109/COMPSAC.2019.10231

Athiththan, K., Rovinsan, S., Sathveegan, S., Gunasekaran, N., Gunawardena, K. S. A. W., & Kasthurirathna, D. (2018). An ontology-based approach to automate the software development process. *Proceedings of the 2018 IEEE 9th International Conference on Information and Automation for Sustainability (ICIAfS)*, 1–6. https://doi.org/10.1109/ICIAFS.2018.8913339

Babaalla, Z., Jakimi, A., Saadane, R., & Chehri, A. (2025). Towards automatic extraction of UML class diagrams: Creation of an annotated dataset for training deep models. *Procedia Computer Science, 270*, 3152–3161. https://doi.org/10.1016/j.procs.2025.09.440

Bik, N., Lucassen, G., & Brinkkemper, S. (2017). Towards a reference method for creating and using textual user stories in software development. *2017 IEEE 25th International Requirements Engineering Workshop (REW)*, 1–8. https://doi.org/10.1109/REW.2017.83

Cohn, M. (2015). Chapter 1 overview. In *Textual user stories applied for agile software development* (pp. 4–16). Addison-Wesley.

Elallaoui, M., Nafil, K., & Touahni, R. (2018). Automatic transformation of user textual user stories into UML use case diagrams using NLP techniques. *Procedia Computer Science, 130*, 42–49. https://doi.org/10.1016/j.procs.2018.04.010

Ferrari, A., Abualhaijal, S., & Arora, C. (2024). Model generation with LLMs: From requirements to UML sequence diagrams. In G. Liebel, I. Hadar, & P. Spoletini (Eds.), *Proceedings of the 32nd IEEE International Requirements Engineering Conference Workshops (REW 2024)* (pp. 291–300). IEEE. https://doi.org/10.1109/REW61692.2024.00044

Fischbach, J., Vogelsang, A., Spies, D., Wehrle, A., Junker, M., & Freudenstein, D. (2020). SPECMATE: Automated creation of test cases from acceptance criteria. *Proceedings of the 2020 IEEE 13th International Conference on Software Testing, Verification and Validation (ICST)*, 321–331. https://doi.org/10.1109/ICST46399.2020.00040

Gilson, F., & Irwin, C. (2018). From textual user stories to use case scenarios: Towards a generative approach. *Proceedings of the 25th Australasian Software Engineering Conference (ASWEC 2018)*, 61–65. https://doi.org/10.1109/ASWEC.2018.00016

Gilson, F., Galster, M., & Georis, F. (2020). Generating use case scenarios from textual user stories. *Proceedings of the 2020 IEEE/ACM International Conference on Software and System Processes (ICSSP)*, 31–40. https://doi.org/10.1145/3379177.3388895

Gunes, T., & Aydemir, F. B. (2020). Automated goal model extraction from textual user stories using NLP. *Proceedings of the IEEE International Conference on Requirements Engineering (RE 2020)*, 382–387. https://doi.org/10.1109/RE48521.2020.00052

Gunes, T., Oz, C. A., & Aydemir, F. B. (2021). ArTu: A tool for generating goal models from textual user stories. *Proceedings of the IEEE International Conference on Requirements Engineering (RE 2021)*, 436–437. https://doi.org/10.1109/RE51729.2021.00058

Jahan, M., Hassan, M. M., Golpayegani, R., Ranjbaran, G., Roy, C., Roy, B., & Schneider, K. (2024). Automated derivation of UML sequence diagrams from user stories: Unleashing the power of generative AI vs. a rule-based approach. *Proceedings of the ACM Symposium on Applied Computing (SAC)*, 138–148. https://doi.org/10.1145/3640310.3674081

Javed, M., & Lin, Y. (2018). Iterative process for generating ER diagram from unrestricted requirements. *Proceedings of the 13th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2018)*, 192–204. https://doi.org/10.5220/0006778701920204

Javed, M., & Lin, Y. (2021). iMER: Iterative process of entity relationship and business process model extraction from the requirements. *Information and Software Technology, 135*, 106558. https://doi.org/10.1016/j.infsof.2021.106558

Kitchenham, B. (2004). *Procedures for performing systematic reviews* (Technical Report TR/SE-0401). Department of Computer Science, Keele University.

Koç, H., Erdoğan, A. M., Barjakly, Y., & Peker, S. (2021). UML diagrams in software engineering research: A systematic literature review. *Proceedings, 74*(1), 13. https://doi.org/10.3390/proceedings2021074013

Kochbati, T., Li, S., Gérard, S., & Mraidha, C. (2021). From textual user stories to models: A machine learning empowered automation. *Proceedings of the 9th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2021)*, 28–40. SCITEPRESS.

Lano, K., Yassipour-Tehrani, S., & Umar, M. A. (2021). Automated requirements formalisation for Agile MDE. *Proceedings of the 24th International Conference on Model-Driven Engineering Languages and Systems Companion (MODELS-C 2021)*, 173–180. https://doi.org/10.1109/MODELS-C53483.2021.00030

Loniewski, G., Insfran, E., & Abrahão, S. (2010). A systematic review of the use of requirements engineering techniques in model-driven development. *Lecture Notes in Computer Science, 6395*, 213–227. https://doi.org/10.1007/978-3-642-16129-2_16

Maatuk, A. M., & Abdelnabi, E. A. (2021). Generating UML use case and activity diagrams using NLP techniques and heuristic rules. *Proceedings of the International Conference on Software and System Processes (ICSSP) – ACM International Conference Proceeding Series*, 271–277. https://doi.org/10.1145/3460620.3460768

Molla, M. M. I., Ahmad, J., & Wan Kadir, W. M. N. (2024). A comparison of transforming the user stories and functional requirements into UML use case diagram. *International Journal of Innovative Computing, 14*(1), 29–36. https://doi.org/10.11113/ijic.v14n1.463

Mustafa, A., Wan-Kadir, W. M. N., Ibrahim, N., & Shah, M. A. (2021). Automated test case generation from requirements: A systematic literature review. *Computers, Materials & Continua, 68*(1), 1703–1724. https://doi.org/10.32604/cmc.2021.014391

Nair, R. P., & Thushara, M. G. (2025). NL2Code: A hybrid NLP and model-driven framework for automated code generation from natural language and UML. *Proceedings of the 2025 IEEE International Students' Conference on Electrical, Electronics and Computer Science (SCEECS)*, 1–6. IEEE. https://doi.org/10.1109/SCEECS64059.2025.10940505

Nasiri, S., Rhazali, Y., Lahmer, M., & Adadi, A. (2021). From textual user stories to UML diagrams driven by ontological and production model. *International Journal of Advanced Computer Science and Applications, 12*(6), 333–340. https://doi.org/10.14569/IJACSA.2021.0120637

Nasiri, S., Rhazali, Y., Lahmer, M., & Chenfour, N. (2020). Towards a generation of class diagram from textual user stories in agile methods. *Procedia Computer Science, 170*, 831–837. https://doi.org/10.1016/j.procs.2020.03.148

Nazir, F., Butt, W. H., Anwar, M. W., & Khan Khattak, M. A. (2017). The applications of natural language processing (NLP) for software requirement engineering: A systematic literature review. *Lecture Notes in Electrical Engineering, 424*, 485–493. https://doi.org/10.1007/978-981-10-4154-9_56

Page, M. J., McKenzie, J. E., Bossuyt, P. M., Boutron, I., Hoffmann, T. C., Mulrow, C. D., Shamseer, L., Tetzlaff, J. M., Akl, E. A., Brennan, S. E., Chou, R., Glanville, J., Grimshaw, J. M., Hróbjartsson, A., Lalu, M. M., Li, T., Loder, E. W., Mayo-Wilson, E., McDonald, S., … Moher, D. (2021). The PRISMA 2020 statement: An updated guideline for reporting systematic reviews. *The BMJ, 372*, n71. https://doi.org/10.1136/bmj.n71

Raharjana, I. K., Siahaan, D., & Fatichah, C. (2021). Textual user stories and natural language processing: A systematic literature review. *IEEE Access, 9*, 53811–53826. https://doi.org/10.1109/ACCESS.2021.3070606

Ramzan, M., Sadiqi, G. S., Bashir, M. S., Raza, S., & Batool, A. (2024). A rule-based approach for automatic generation of class diagram from functional requirements using natural language processing and machine learning. *Journal of Computing & Biomedical Informatics, 7*(2). https://doi.org/10.56979/702/2024

Schlutter, A., & Vogelsang, A. (2020). Knowledge extraction from natural language requirements into a semantic relation graph. *Proceedings of the 2020 IEEE/ACM 42nd International Conference on Software Engineering Workshops (ICSEW)*, 373–379. https://doi.org/10.1145/3387940.3392162

Seidl, M., Scholz, M., Huemer, C., & Kappel, G. (2015). The use case diagram. In *UML @ Classroom: An Introduction to Object-Oriented Modeling* (pp. 23–47). Springer. https://doi.org/10.1007/978-3-319-12742-2_3

Shweta, Sanyal, R., & Ghoshal, B. (2018). Automatic extraction of structural model from semi-structured software requirement specification. *Proceedings of the 17th IEEE/ACIS International Conference on Computer and Information Science (ICIS 2018)*, 543–548. https://doi.org/10.1109/ICIS.2018.8466406

Siddeshwar, V. (2025). *Automated goal model generation from user stories using large language models* (Master's thesis, Ontario Tech University). Ontario Tech Scholaris. https://ontariotechu.scholaris.ca/server/api/core/bitstreams/df4c51d3-55a2-42ca-a24f-28992eb3687f/content

Tejaswini, N., Sharath, S. H., Sanjitha, A., Thomas, S., Varshini, A., & Ghogge, R. (2025). UMLify: Instant UML diagrams from scenario descriptions. *Proceedings of the 2025 International Conference on Data Science, Agents & Artificial Intelligence (ICDSAAI)*, 1–5. IEEE. https://doi.org/10.1109/ICDSAAI65575.2025.11011782

Ternes, B., Rosenthal, K., & Strecker, S. (2021). Automated assistance for data modelers combining natural language processing and data modeling heuristics: A prototype demonstration. *CEUR Workshop Proceedings, 2958*, 25–30.

Thesing, T. C. F. M. B. (2021). Agile versus waterfall project management: Decision model for selecting the appropriate approach to a project. *Procedia Computer Science, 181*, 746–756. https://doi.org/10.1016/j.procs.2021.01.227

Tiwari, S., Ameta, D., & Banerjee, A. (2019). An approach to identify use case scenarios from textual requirements specification. *Proceedings of the 2019 International*

Conference on Software Engineering and Knowledge Engineering (ICSEKE) – ACM International Conference Proceeding Series, 1–8. https://doi.org/10.1145/3299771.3299774

Vasques, D. G., Santos, G. S., Gomes, F. D., Galindo, J. F., & Martins, P. S. (2019). Use case extraction through knowledge acquisition. *Proceedings of the 2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON 2019)*, 624–631. https://doi.org/10.1109/IEMCON.2019.8936279

Zhao, L., Alhoshan, W., Ferrari, A., Letsholo, K. J., Ajagbe, M. A., Chioasca, E.-V., & Batista-Navarro, R. T. (2021). Natural language processing for requirements engineering: A systematic mapping study. *ACM Computing Surveys, 54*(3), 1–41. https://doi.org/10.1145/3444689

APPENDICES

Appendix A: Journal Publications

1. **Mornie, M.N**., Jali, N., Junaini, S.N., Mit, E., Shiang, C.W., & Saee, S. (2023). Visualisation of Textual user stories in UML Models: A Systematic Literature Review. *Acta Informatica Pragensia*, *12*(2), 419-438. doi: 10.18267/j.aip.212
2. **Mornie, M. N.,** Jali, N., Cheah, W. S., Mit, E., Saee, S., Jali, S. K., & Greer, D. (2025). Visualisation of Textual user stories to UML use Case Diagram. *Journal of Advanced Research in Applied Sciences and Engineering Technology*, *63*(3), 68–80. https://doi.org/10.37934/araset.63.3.6880

Appendix B: Survey Questions for Conducting Preliminary Studies

Google Forms link to the survey questions:

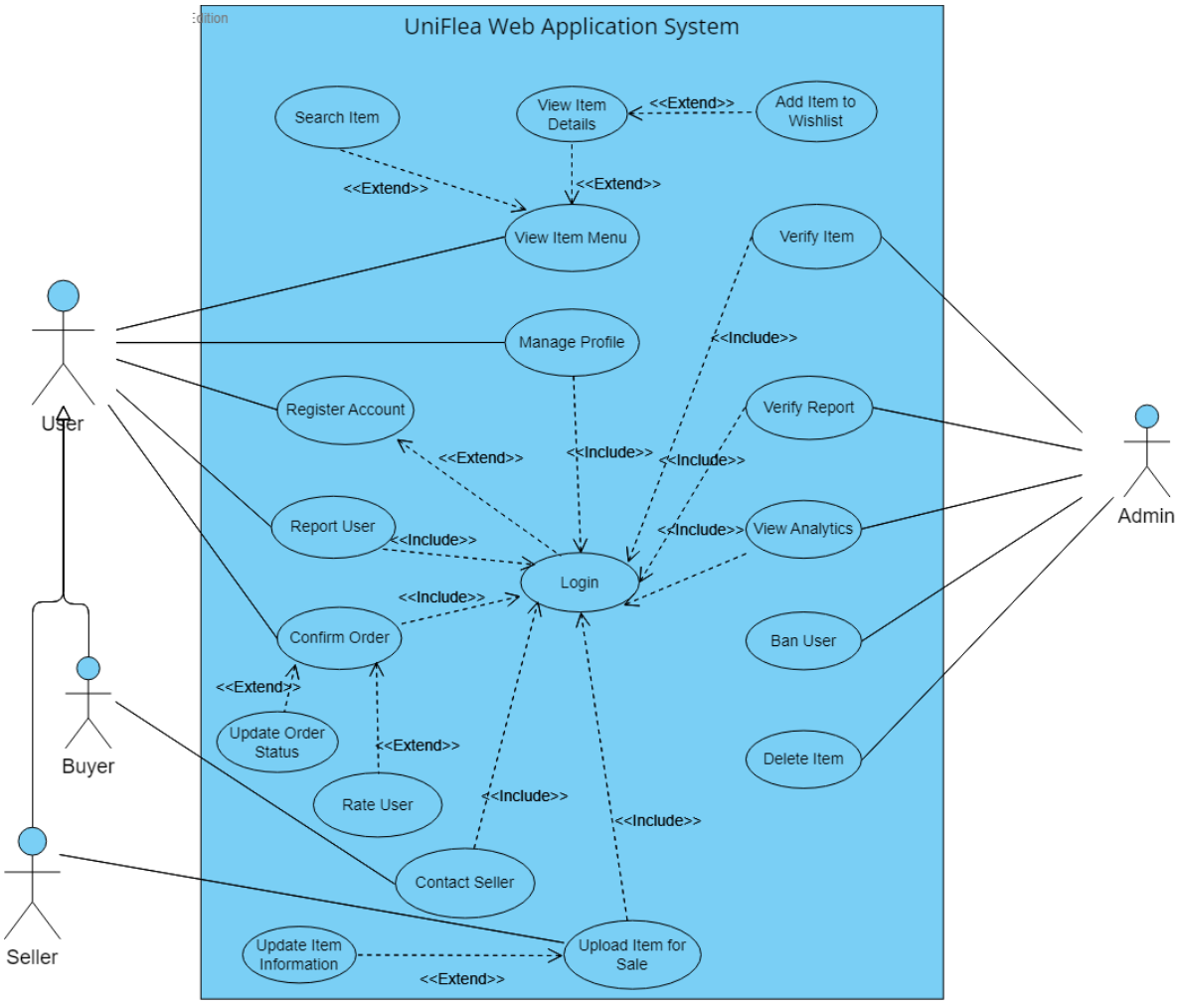
https://docs.google.com/forms/d/1owsJBG69179CiwADO91EKoPuz36rNyH0ayBLwi6C2Ng

Appendix C: Collected Textual user stories and UML Use Case Diagram Samples

1. University Flea Market Web Application System

A platform that facilitates online buying and selling for users. It includes functionalities like searching items, managing profiles, viewing analytics, verifying items, and managing orders.

a. Use case diagram



b. Textual user stories

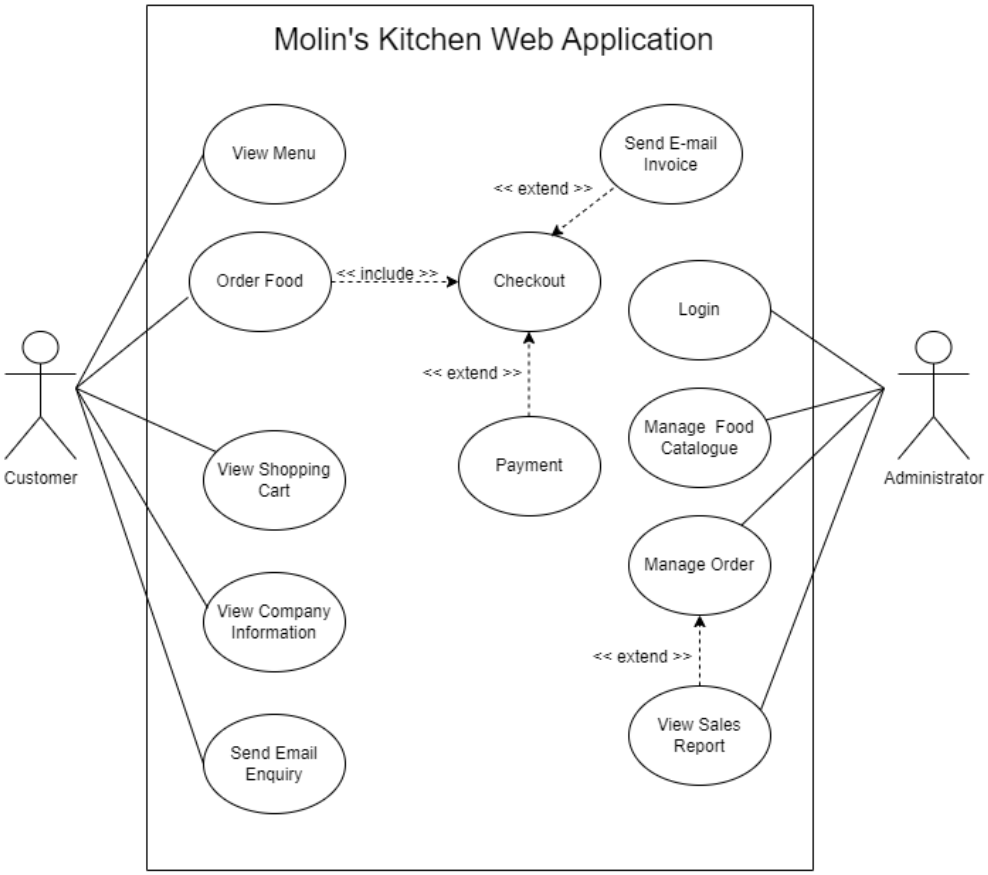
No	Textual user stories
1	As a buyer, I want to confirm my order so that I can proceed with the purchase.
2	As a buyer, I want to contact the seller so that I can ask questions about the item.
3	As a buyer, I want to rate users so that I can share feedback about my experience.
4	As a seller, I want to upload items for sale so that I can list them on the platform.
5	As a seller, I want to update item information so that I can provide accurate details to buyers.
6	As a seller, I want to update the order status so that buyers are informed about their purchase.
7	As an admin, I want to verify uploaded items so that only appropriate content is listed.
8	As an admin, I want to verify user reports so that I can take necessary actions against violations.
9	As an admin, I want to view analytics so that I can monitor platform performance and user activity.
10	As an admin, I want to ban users who violate policies so that the platform remains safe.
11	As an admin, I want to delete inappropriate or prohibited items so that the platform adheres to its standards.

12	As a user, I want to search for items so that I can quickly find what I need.
13	As a user, I want to browse the item menu so that I can explore available products.
14	As a user, I want to view detailed information about an item so that I can make an informed decision.
15	As a user, I want to add items to my wishlist so that I can save them for later.
16	As a user, I want to register for an account so that I can access the platform features.
17	As a user, I want to log in securely so that I can access my account.
18	As a user, I want to manage my profile so that I can update my personal details and preferences.
19	As a user, I want to report other users so that I can flag inappropriate behaviour or content.

Molin’s Kitchen Web Application

A web-based food ordering system that allow online transactions between customers and seller replacing conventional food ordering method while also reducing the need for manually gathering order from the customer.

a. Use case diagram



b. Textual user stories

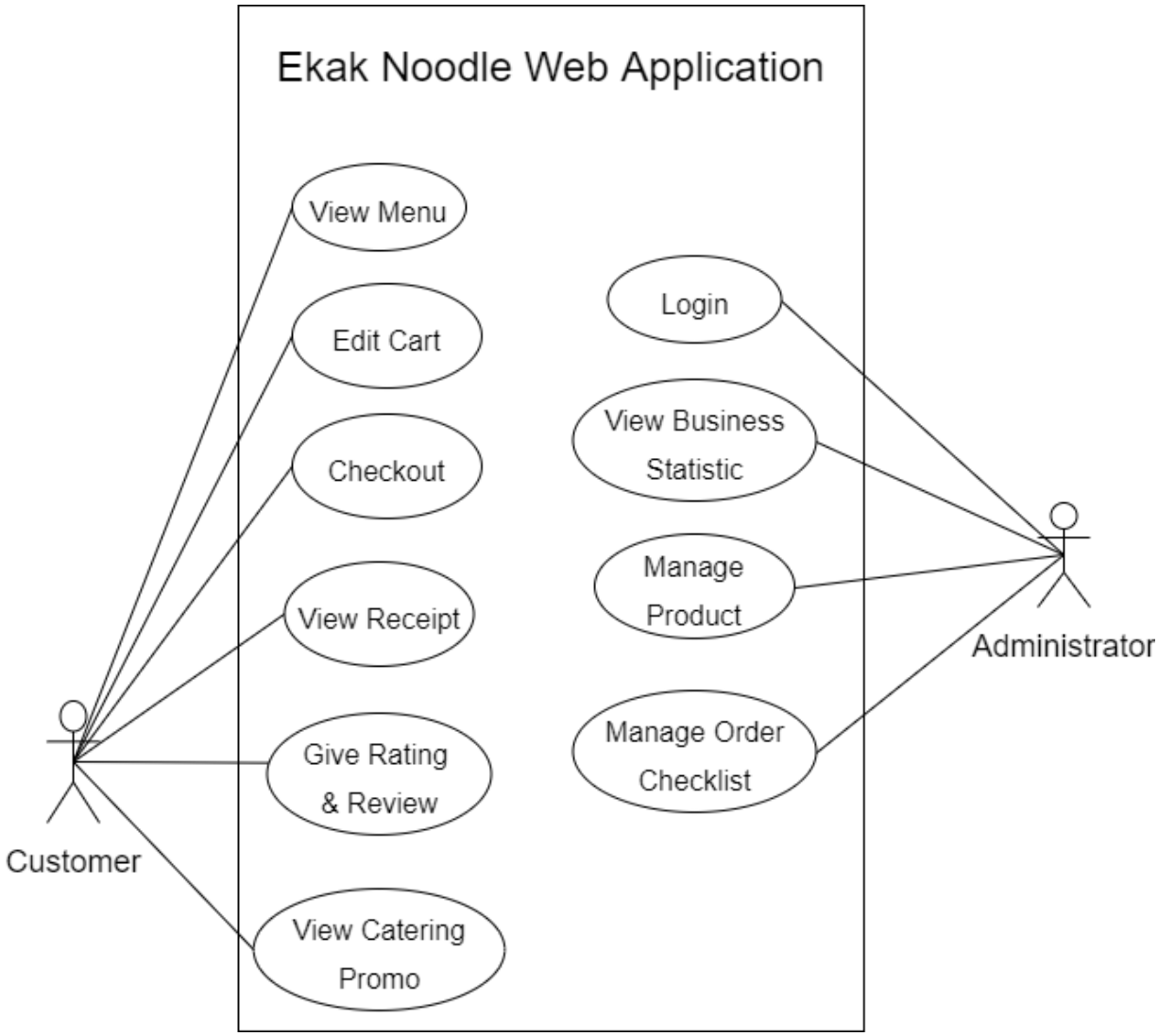
No.	Textual user stories
1	As a customer, I want to view the menu so that I can decide what food to order.
2	As a customer, I want to order food so that I can purchase items from the application.

3	As a customer, I want to view my shopping cart so that I can review and edit my order.
4	As a customer, I want to view company information so that I can learn more about the business.
5	As a customer, I want to send an email inquiry so that I can ask questions or provide feedback.
6	As a customer, I want to proceed to checkout so that I can complete my order.
7	As a customer, I want to make a payment so that I can finalise my purchase.
8	As a customer, I want to receive an email invoice so that I have a record of my purchase.
9	As an administrator, I want to log in so that I can access administrative features.
10	As an administrator, I want to manage the food catalogue so that I can update the menu offerings.
11	As an administrator, I want to manage orders so that I can track and fulfil customer requests.
12	As an administrator, I want to view sales reports so that I can monitor business performance.

Ekak Noodle Web Application

A food ordering web application for customers to view menus, edit carts, checkout, and view receipts. It also comes with admin features like business statistics and product management.

a. Use case diagram



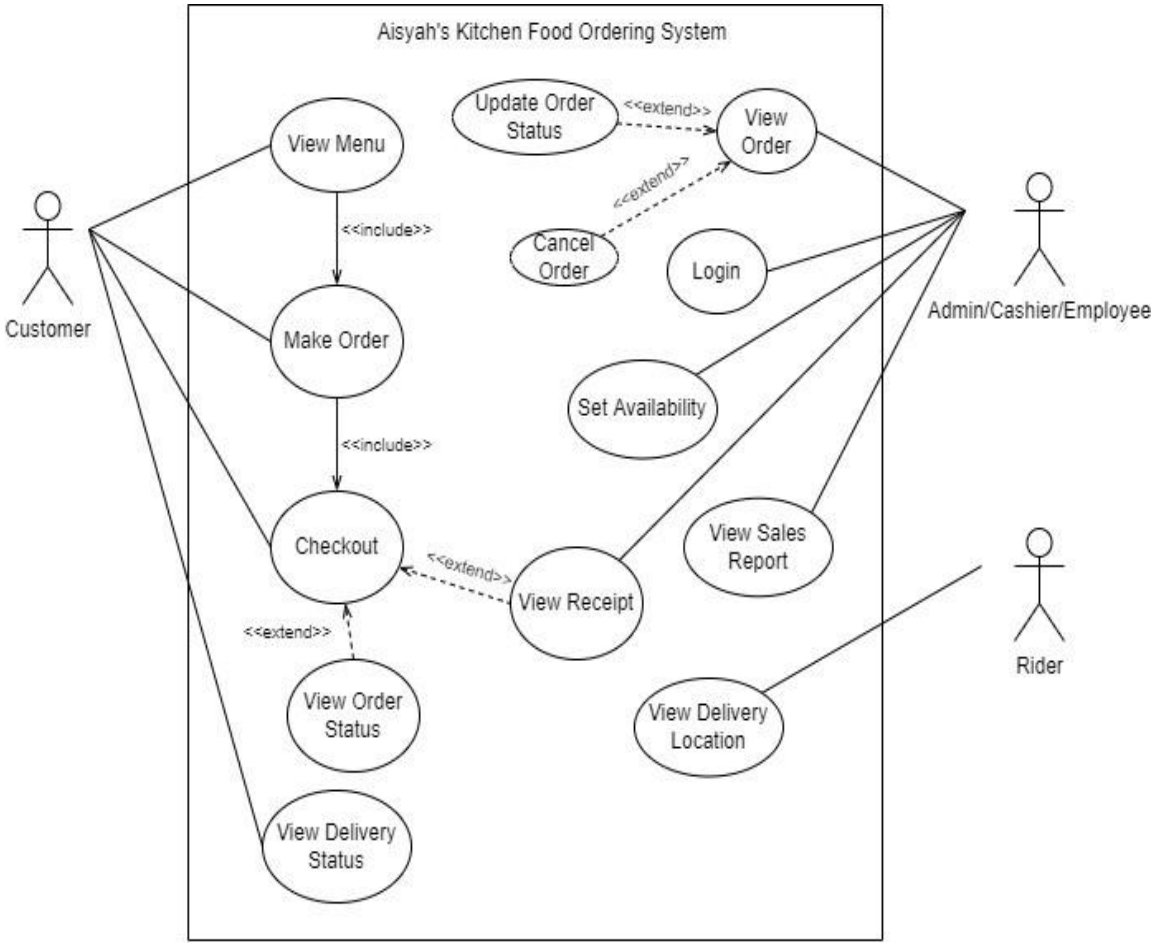
b. Textual user stories

No.	Textual user stories
1	As a customer, I want to view the menu so that I can explore available dishes.
2	As a customer, I want to edit my cart so that I can update my selected items before checkout.
3	As a customer, I want to proceed to checkout so that I can complete my purchase.
4	As a customer, I want to view my receipt so that I can review the details of my order.
5	As a customer, I want to give ratings and reviews so that I can share feedback about my experience.
6	As a customer, I want to view catering promotions so that I can take advantage of special deals.
7	As an administrator, I want to log in so that I can access administrative features.
8	As an administrator, I want to view business statistics so that I can monitor performance.
9	As an administrator, I want to manage products so that I can update and maintain the menu offerings.
10	As an administrator, I want to manage the order checklist so that I can track and fulfill customer orders.

Aisyah’s Kitchen Food Ordering System

A food ordering system where customers can view menus, make orders, track order status, and checkout, while admins manage orders and generate sales reports.

a. Use case diagram



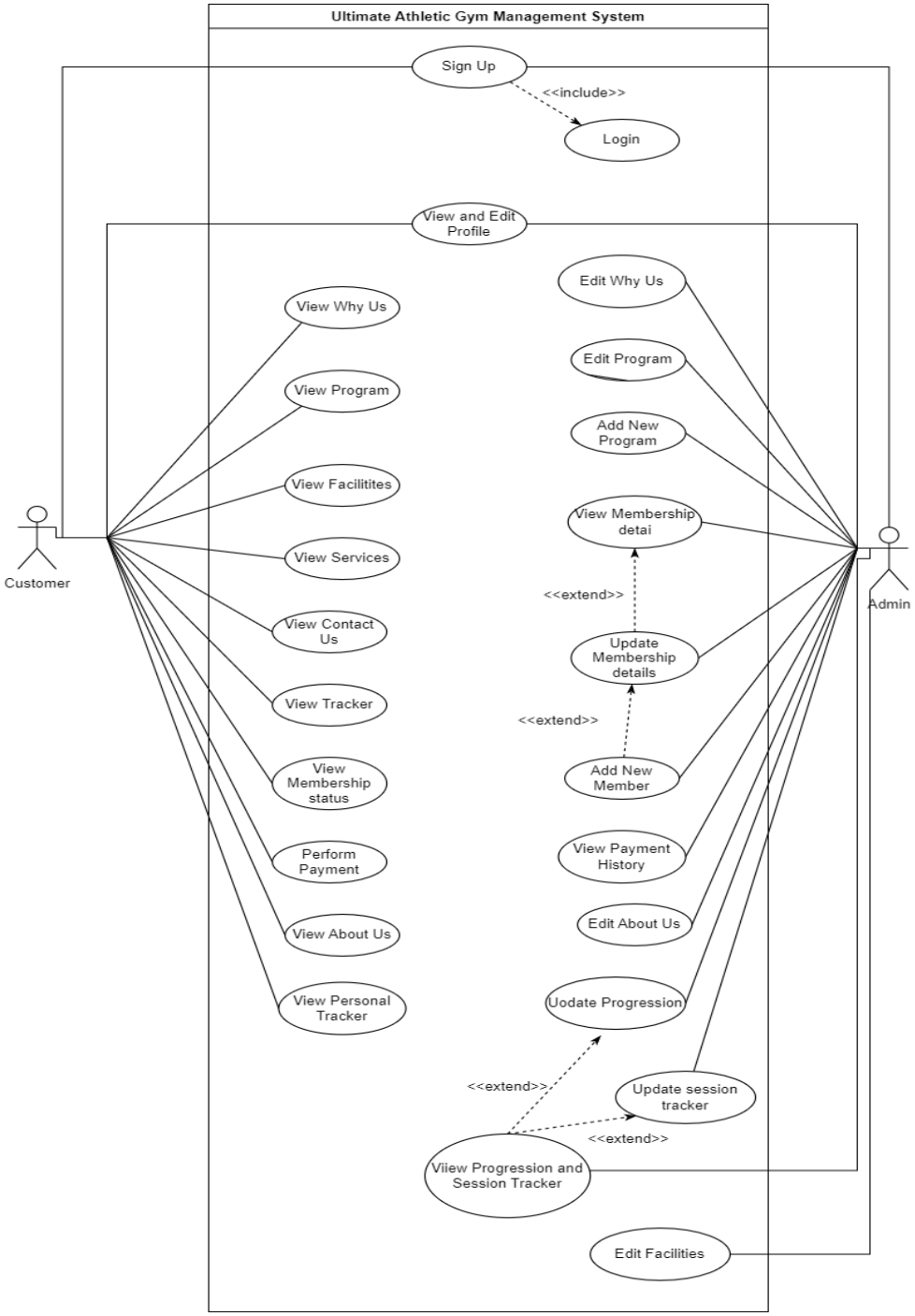
b. Textual user stories

No.	Textual user stories
1	As a customer, I want to view the menu so that I can decide what to order.
2	As a customer, I want to make an order so that I can purchase food from the system.
3	As a customer, I want to proceed to checkout so that I can complete my order.
4	As a customer, I want to view the status of my order so that I can track its progress.
5	As a customer, I want to view the delivery status so that I know when my order will arrive.
6	As a customer, I want to view my receipt so that I can confirm the details of my transaction.
7	As an admin, I want to log in so that I can access the system and perform administrative tasks.
8	As an admin, I want to view orders so that I can monitor customer requests.
9	As an admin, I want to update the status of an order so that customers are informed of its progress.
10	As an admin, I want to cancel an order so that I can handle issues or errors in the process.
11	As an admin, I want to set availability so that I can manage which items are currently offered.
12	As an admin, I want to view sales reports so that I can monitor business performance.
13	As a rider, I want to view the delivery location so that I can deliver the order to the customer.

Ultimate Athletic Gym Management System

A gym management platform enabling customers to view programs, facilities, and personal trackers while admins manage memberships, programs, and gym facilities.

a. Use case diagram



b. Textual user stories

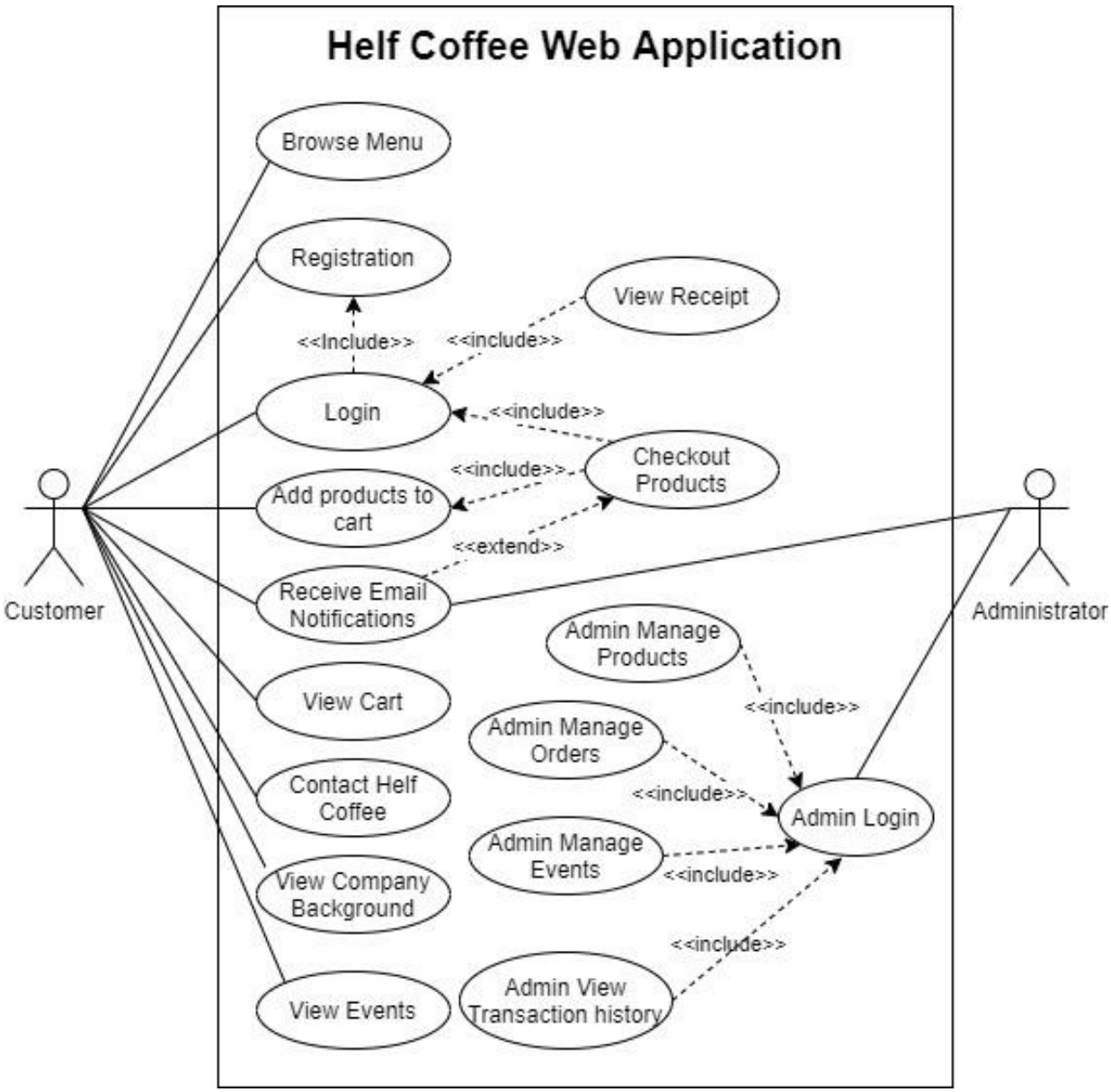
No.	Textual user stories
1	As a customer, I want to sign up so that I can create an account.
2	As a customer, I want to log in so that I can access my account.
3	As a customer, I want to view and edit my profile so that I can keep my information updated.
4	As a customer, I want to view programs so that I can explore available fitness programs.
5	As a customer, I want to view facilities so that I can see what amenities are available.
6	As a customer, I want to view services so that I can know what the gym offers.
7	As a customer, I want to view the "Why Us" section so that I can understand the benefits of the gym.
8	As a customer, I want to view the tracker so that I can monitor my fitness progress.
9	As a customer, I want to view my membership status so that I can check my subscription details.
10	As a customer, I want to perform payments so that I can renew or pay for my membership.
11	As a customer, I want to view the "About Us" section so that I can learn more about the gym.
12	As a customer, I want to view my personal tracker so that I can track my individual progress.
13	As a customer, I want to view progression and session trackers so that I can analyse my performance.
14	As an admin, I want to edit the "Why Us" section so that I can keep it relevant and updated.

15	As an admin, I want to edit programs so that I can modify the gym's fitness offerings.
16	As an admin, I want to add new programs so that I can expand the gym's offerings.
17	As an admin, I want to view membership details so that I can oversee customers' memberships.
18	As an admin, I want to update membership details so that I can correct or change customer records.
19	As an admin, I want to add new members so that I can register new customers.
20	As an admin, I want to view payment history so that I can track customers' transactions.
21	As an admin, I want to edit the “About Us” section so that I can keep it current and accurate.
22	As an admin, I want to update progression so that I can maintain accurate fitness progress data.
23	As an admin, I want to update session trackers so that I can manage session records.
24	As an admin, I want to edit facilities so that I can manage and update facility details.

Helf Coffee Website System

A coffee ordering platform for customers to browse menus, place orders, checkout, and inquire, while admins manage events, orders, and products.

a. Use case diagram



b. Textual user stories

No.	Textual user stories
1	As a customer, I want to browse the menu so that I can explore available coffee products.
2	As a customer, I want to register so that I can create an account for purchases.
3	As a customer, I want to log in so that I can access my account and personalised features.
4	As a customer, I want to add products to my cart so that I can prepare my order for checkout.
5	As a customer, I want to view my cart so that I can review and modify the items before purchase.
6	As a customer, I want to checkout products so that I can complete my order.
7	As a customer, I want to view my receipt so that I can confirm my order details.
8	As a customer, I want to receive email notifications so that I can stay updated about my orders.
9	As a customer, I want to contact Helf Coffee so that I can ask questions or provide feedback.
10	As a customer, I want to view the company background so that I can learn more about Helf Coffee.
11	As a customer, I want to view events so that I can participate in community activities.
12	As an administrator, I want to log in so that I can access administrative functions.
13	As an administrator, I want to manage products so that I can update the menu offerings.

14	As an administrator, I want to manage orders so that I can track and fulfill customer requests.
15	As an administrator, I want to manage events so that I can organise community activities.
16	As an administrator, I want to view transaction history so that I can monitor financial records.

Serene Housing Management System

A housing management system where customers can view properties, book appointments, and receive confirmations, while admins oversee bookings, generate reports, and manage appointments.

a. Use case diagram

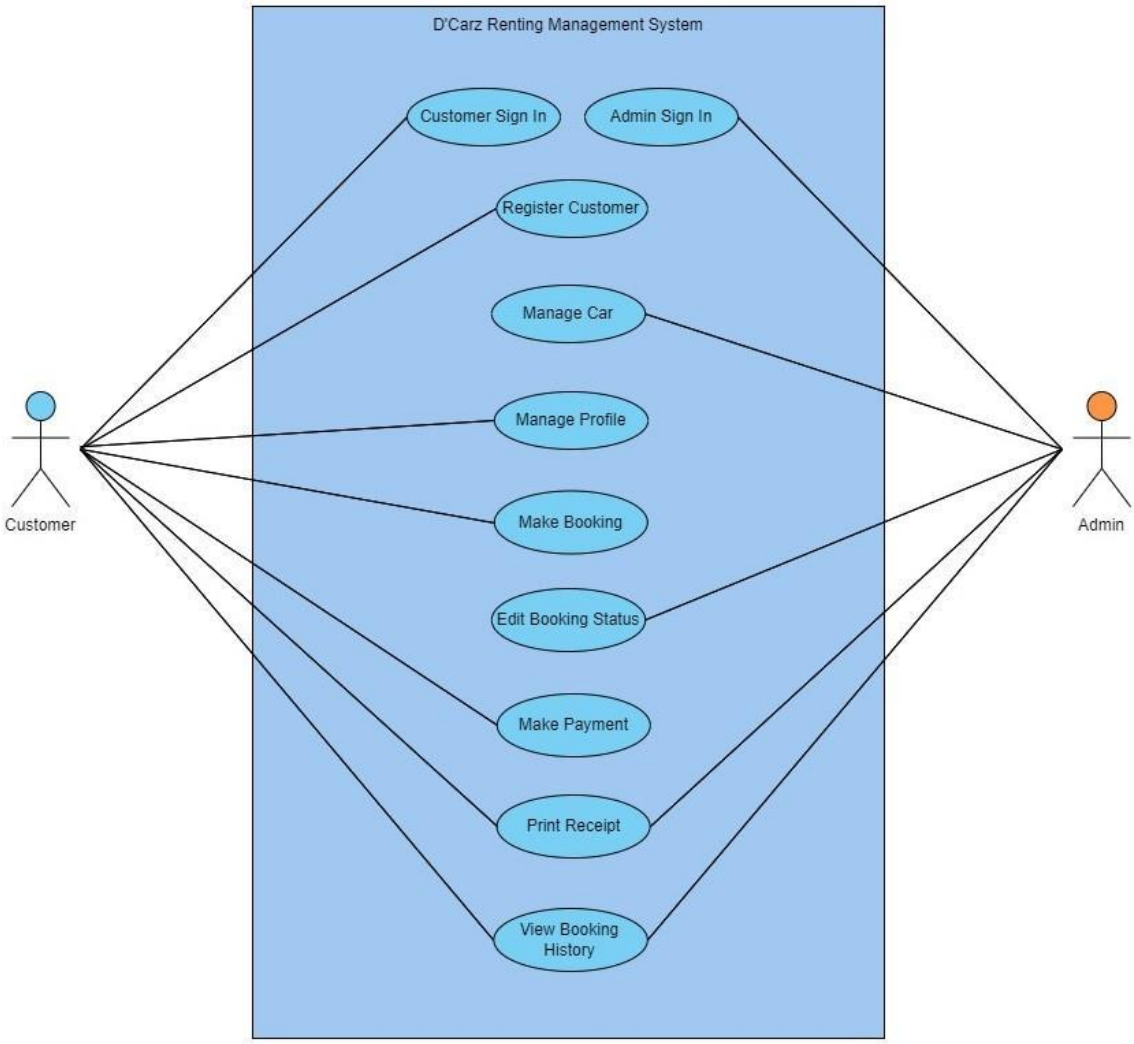
b. Textual user stories

No.	Textual user stories
1	As a customer, I want to log in so that I can access my account.
2	As a customer, I want to view properties so that I can explore available housing options.
3	As a customer, I want to book a property so that I can secure a reservation.
4	As a customer, I want to receive a booking confirmation email so that I can have proof of my booking.
5	As a customer, I want to view the company description so that I can learn more about the organisation.
6	As a customer, I want to view property descriptions so that I can get detailed information about listings.
7	As a customer, I want to send an email inquiry so that I can ask questions or request more information.
8	As an admin, I want to log in so that I can access administrative features.
9	As an admin, I want to view the admin dashboard so that I can manage the system effectively.
10	As an admin, I want to view bookings so that I can track customer reservations.
11	As an admin, I want to view appointments so that I can manage scheduled meetings.
12	As an admin, I want to set appointments so that I can schedule meetings with customers.
13	As an admin, I want to generate reports so that I can analyse booking and appointment data.

D’Carz Renting Management System

A car rental management system where customers can sign in, make bookings, and view booking history, while admins handle car management and booking status.

a. Use case diagram



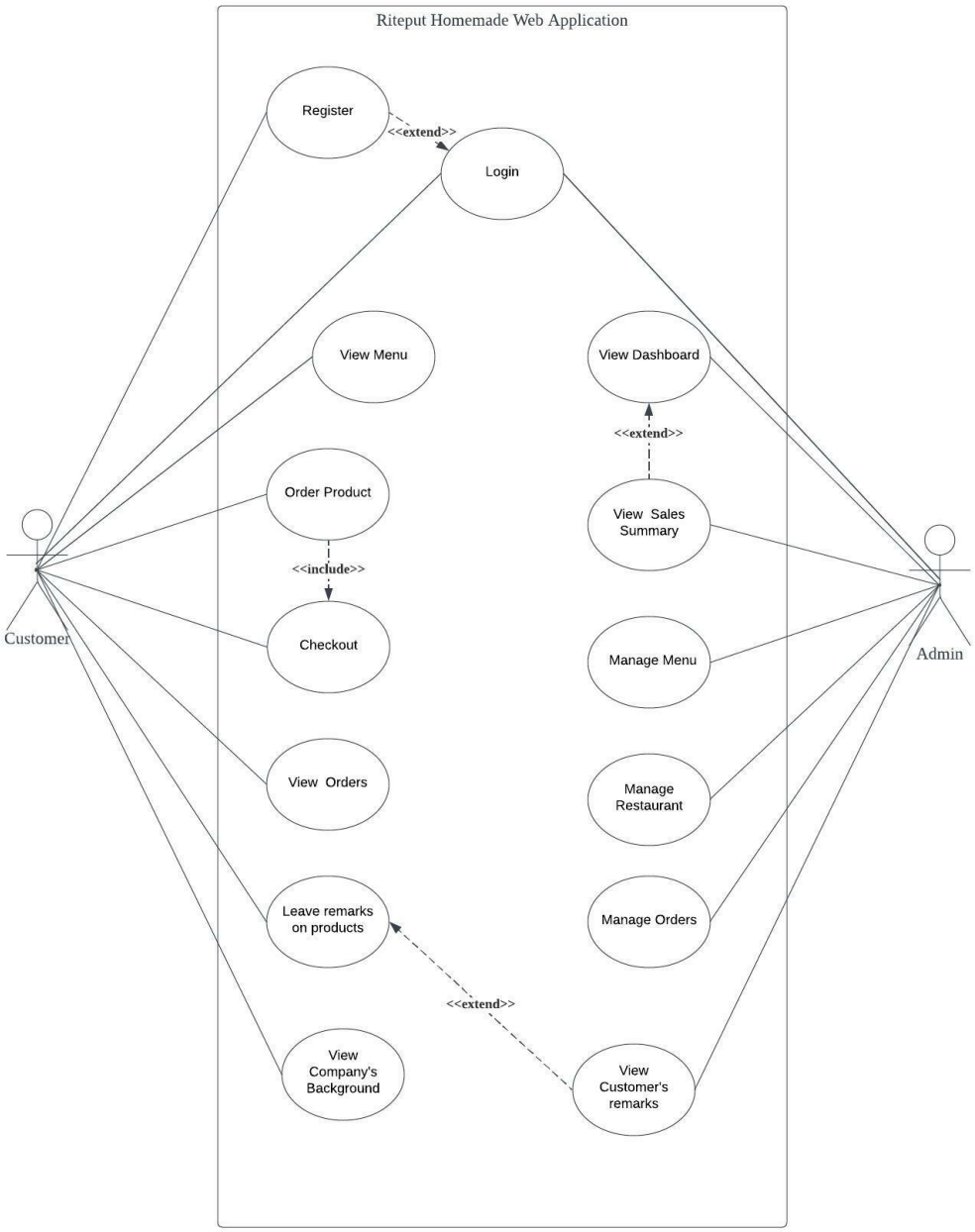
b. Use stories

No.	Textual user stories
1	As a customer, I want to sign in so that I can access my account and services.
2	As a customer, I want to register so that I can create an account to use the system.
3	As a customer, I want to manage my profile so that I can update my personal details.
4	As a customer, I want to make a booking so that I can rent a car.
5	As a customer, I want to make a payment so that I can confirm my booking.
6	As a customer, I want to print a receipt so that I can have proof of my transaction.
7	As a customer, I want to view my booking history so that I can track my past transactions.
8	As an admin, I want to sign in so that I can access administrative functions.
9	As an admin, I want to register customers so that I can add new users to the system.
10	As an admin, I want to manage cars so that I can update and maintain the available vehicles.
11	As an admin, I want to edit booking statuses so that I can keep track of reservations.

Riteput Homemade Web Application

A homemade food ordering platform for customers to view menus, place orders, and leave remarks, with admin capabilities for menu and order management.

a. Use case diagram



Textual user stories

No.	Textual user stories
1	As a customer, I want to register so that I can create an account to use the application.
2	As a customer, I want to log in so that I can access my account and services.
3	As a customer, I want to view the menu so that I can explore available homemade products.
4	As a customer, I want to order a product so that I can purchase items from the menu.
5	As a customer, I want to proceed to checkout so that I can confirm my order.
6	As a customer, I want to view my orders so that I can keep track of my purchases.
7	As a customer, I want to leave remarks on products so that I can share feedback about my experience.
8	As a customer, I want to view the company's background so that I can learn more about its story.
9	As an admin, I want to log in so that I can access administrative features.
10	As an admin, I want to view the dashboard so that I can monitor and manage the application.
11	As an admin, I want to view the sales summary so that I can analyse business performance.
12	As an admin, I want to manage the menu so that I can update available products.
13	As an admin, I want to manage the restaurant so that I can oversee its operations.
14	As an admin, I want to manage orders so that I can track and fulfill customer requests.
15	As an admin, I want to view customer remarks so that I can understand customer feedback.

Shop Valley Website System

An e-commerce platform allowing customers to browse items, add to cart, and checkout, while admins manage inventory and generate sales reports.

a. Use case diagram

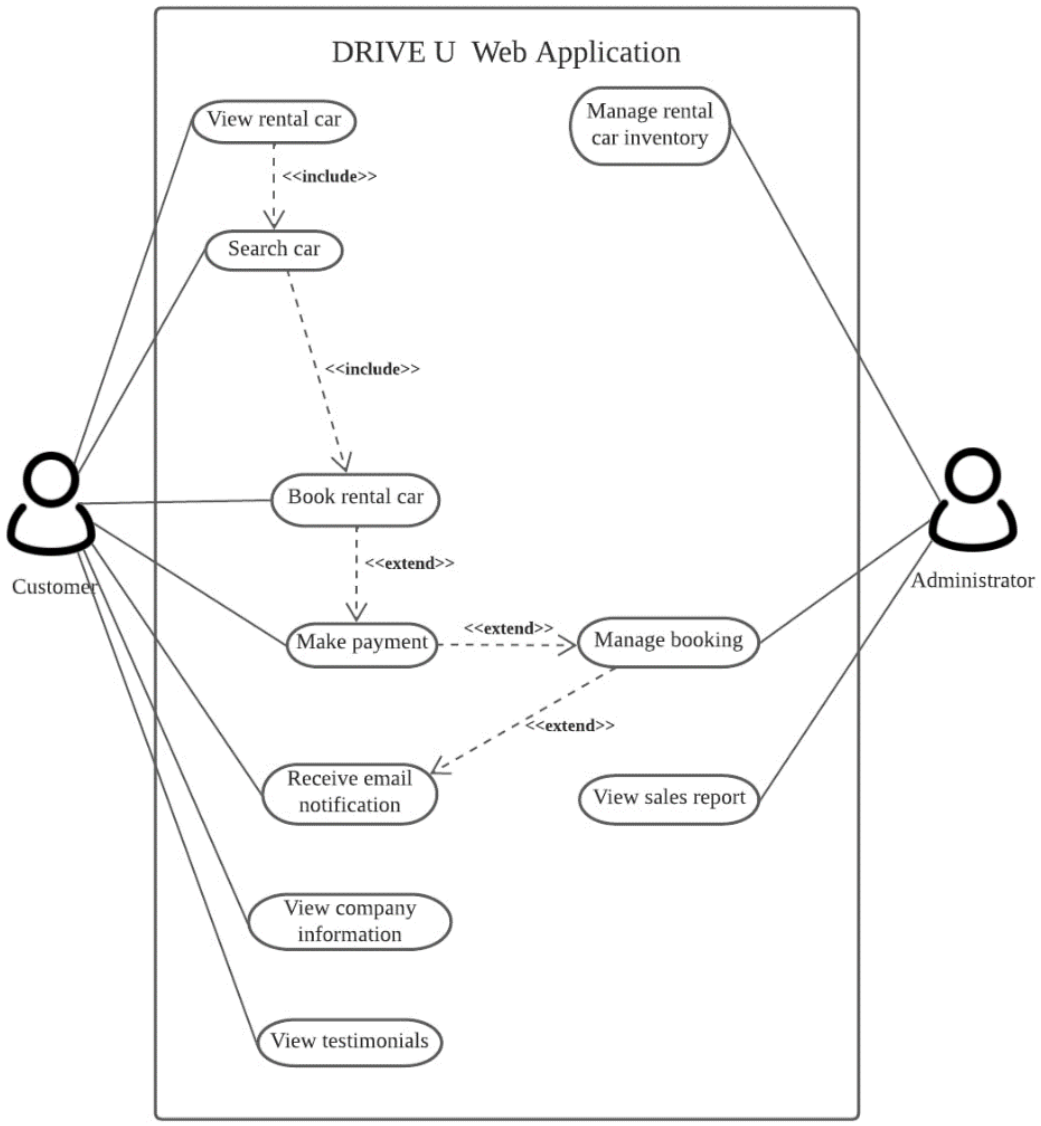
b. Textual user stories

No.	Textual user stories
1	As a customer, I want to register so that I can create an account to use the system.
2	As a customer, I want to log in so that I can access my account and services.
3	As a customer, I want to browse items so that I can explore available products.
4	As a customer, I want to add items to my cart so that I can prepare my order.
5	As a customer, I want to view my shopping cart so that I can review and edit my selected items.
6	As a customer, I want to checkout items so that I can complete my purchase.
7	As a customer, I want to make a payment so that I can confirm my order.
8	As a customer, I want to receive an e-receipt so that I can have a record of my transaction.
9	As a customer, I want to receive an email notification so that I can stay informed about my order.
10	As a customer, I want to view Shop Valley information so that I can learn more about the store.
11	As a customer, I want to view my user profile so that I can check my account details.
12	As a customer, I want to edit my user profile so that I can update my personal information.
13	As an admin, I want to view items inventory so that I can track product availability.
14	As an admin, I want to view the sales report so that I can monitor business performance.
15	As an admin, I want to manage items inventory so that I can update and maintain stock details.

DRIVE U Web Application

A car rental service where customers can view cars, book rentals, and make payments, while admins can view reports, handle inventory and bookings.

a. Use case diagram



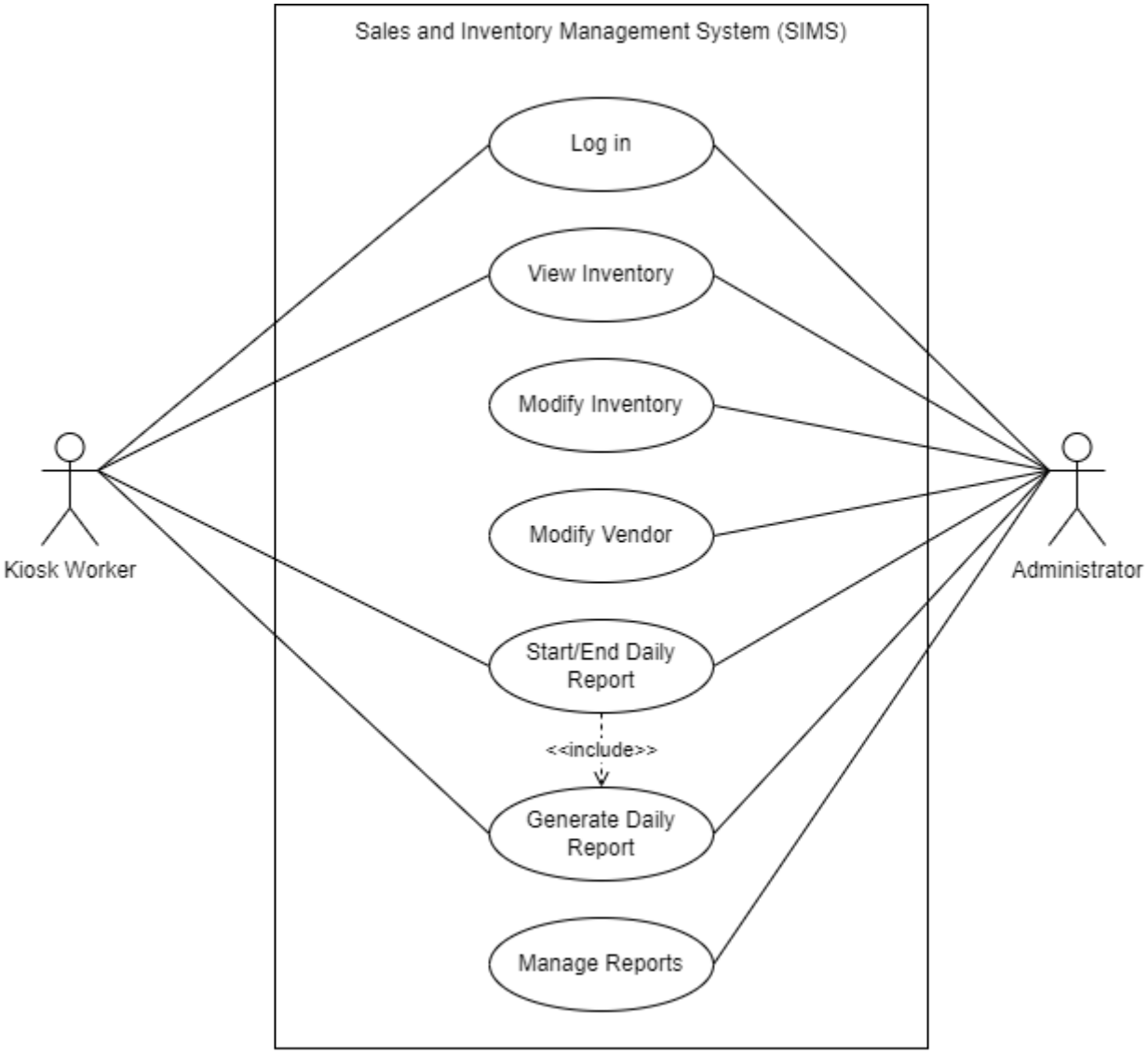
b. Textual user stories

No.	Textual user stories
1	As a customer, I want to view rental cars so that I can explore available vehicles for rent.
2	As a customer, I want to search for a car so that I can find a vehicle that meets my needs.
3	As a customer, I want to book a rental car so that I can reserve it for my use.
4	As a customer, I want to make a payment so that I can confirm my rental booking.
5	As a customer, I want to receive email notifications so that I can stay updated about my bookings.
6	As a customer, I want to view company information so that I can learn more about the service provider.
7	As a customer, I want to view testimonials so that I can read reviews from other users.
8	As an administrator, I want to manage the rental car inventory so that I can update and maintain the fleet.
9	As an administrator, I want to manage bookings so that I can track and handle customer reservations.
10	As an administrator, I want to view sales reports so that I can analyse business performance.

Sales and Inventory Management System (SIMS)

SIMS helps kiosk workers and administrators manage inventory, vendors, and daily operations. Kiosk workers handle stock updates and generate reports, while administrators oversee inventory, vendors, and system records.

a. Use case diagram



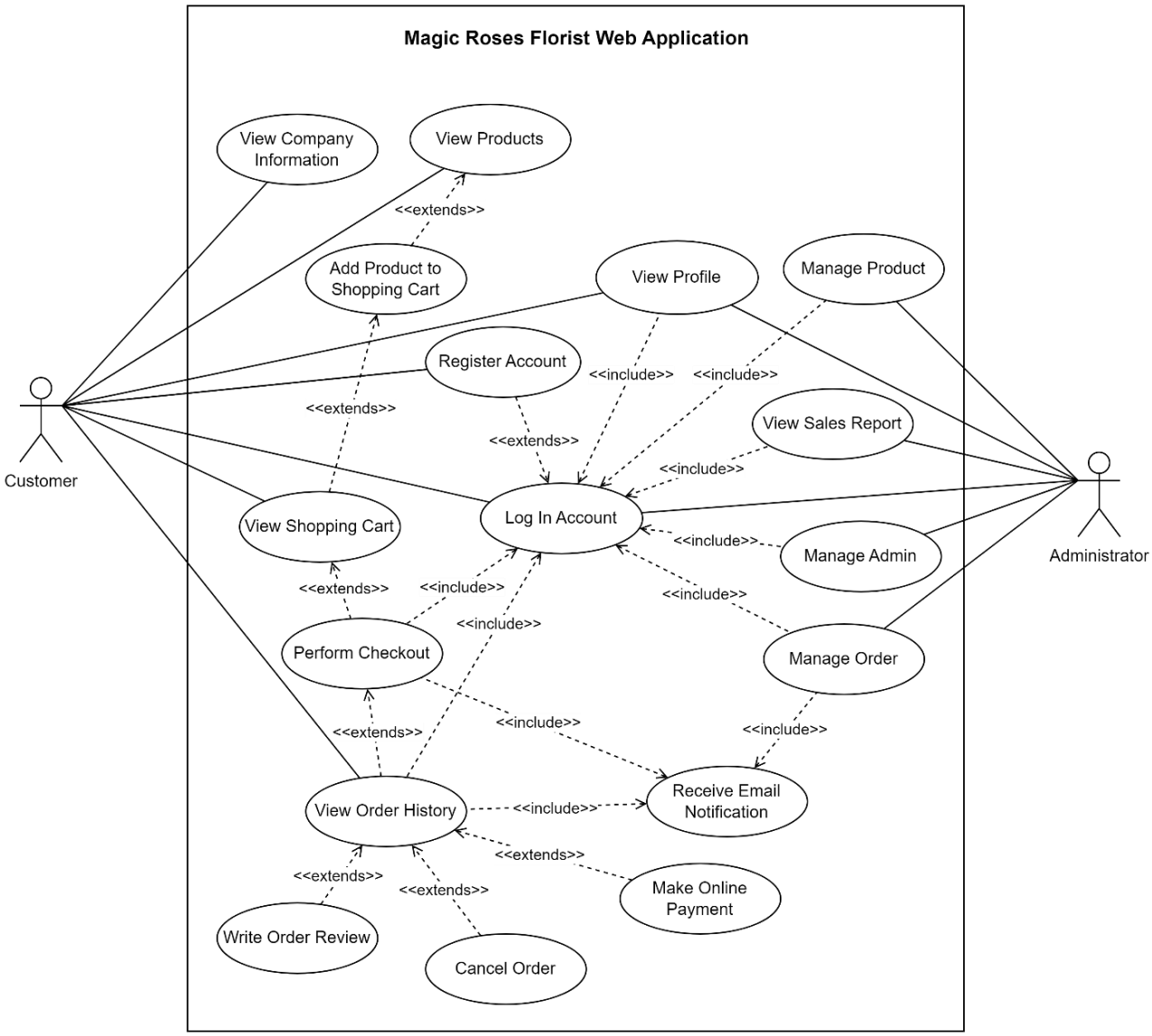
b. Textual user stories

No.	Textual user stories
1	As a kiosk worker, I want to log in so that I can access the system features.
2	As a kiosk worker, I want to view the inventory so that I can check product availability.
3	As a kiosk worker, I want to modify inventory so that I can update stock information.
4	As a kiosk worker, I want to start or end a daily report so that I can track daily activities.
5	As a kiosk worker, I want to generate daily reports so that I can summarise the daily transactions.
6	As an admin, I want to log in so that I can manage administrative tasks.
7	As an admin, I want to view the inventory so that I can monitor stock levels.
8	As an admin, I want to modify the inventory so that I can manage the products available.
9	As an admin, I want to modify vendor information so that I can keep supplier details updated.
10	As an admin, I want to manage reports so that I can review and organise system records.

Magic Roses Florist Web Application

A florist web application that allows customers to browse flower arrangements, place orders, and make payments. Admins can manage product inventory, orders, and generate sales reports.

a. Use case diagram



b. Textual user stories

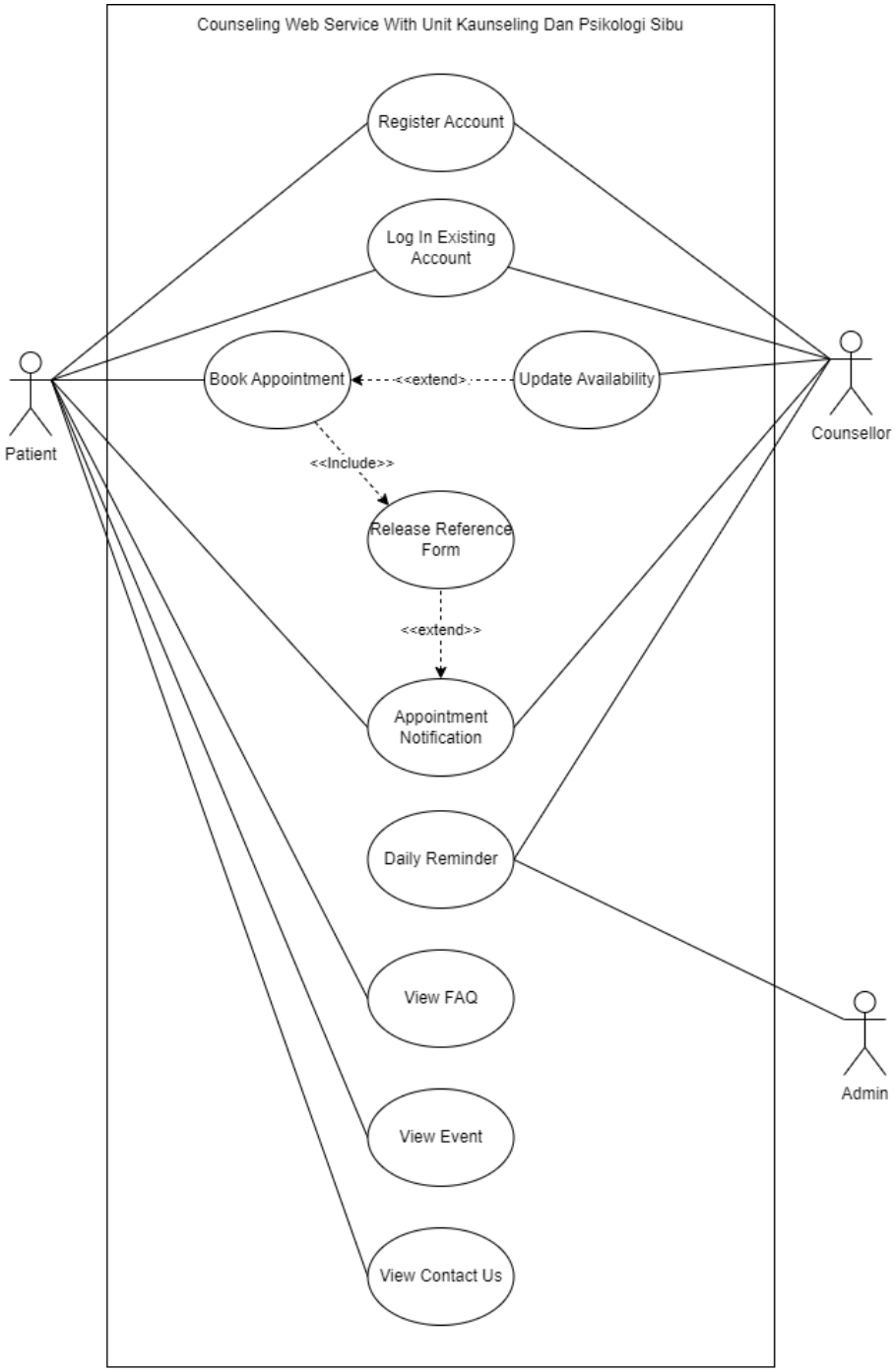
No.	Textual user stories
1	As a customer, I want to view company information so that I can learn more about the business.
2	As a customer, I want to view products so that I can browse available items for purchase.
3	As a customer, I want to add products to my shopping cart so that I can prepare my order.
4	As a customer, I want to register an account so that I can use the system's features.
5	As a customer, I want to log in to my account so that I can access personalised features.
6	As a customer, I want to view my shopping cart so that I can review items before checkout.
7	As a customer, I want to perform checkout so that I can complete my purchase.
8	As a customer, I want to make an online payment so that I can confirm my order.
9	As a customer, I want to view my order history so that I can keep track of my past purchases.
10	As a customer, I want to write an order review so that I can share my feedback about the product.
11	As a customer, I want to cancel an order so that I can stop an unwanted transaction.
12	As a customer, I want to receive email notifications so that I can stay updated about my orders.
13	As a customer, I want to view my profile so that I can check my account details.
14	As an admin, I want to manage products so that I can update the available inventory.
15	As an admin, I want to view sales reports so that I can analyse business performance.

16	As an admin, I want to manage admins so that I can assign and control administrative tasks.
17	As an admin, I want to manage orders so that I can track and fulfill customer requests.

Counselling Web Service with Unit Kaunseling Dan Psikologi Sibu

A counselling platform where patients book appointments and access support, counsellors update availability, and admins manage FAQs and events.

a. Use case diagram



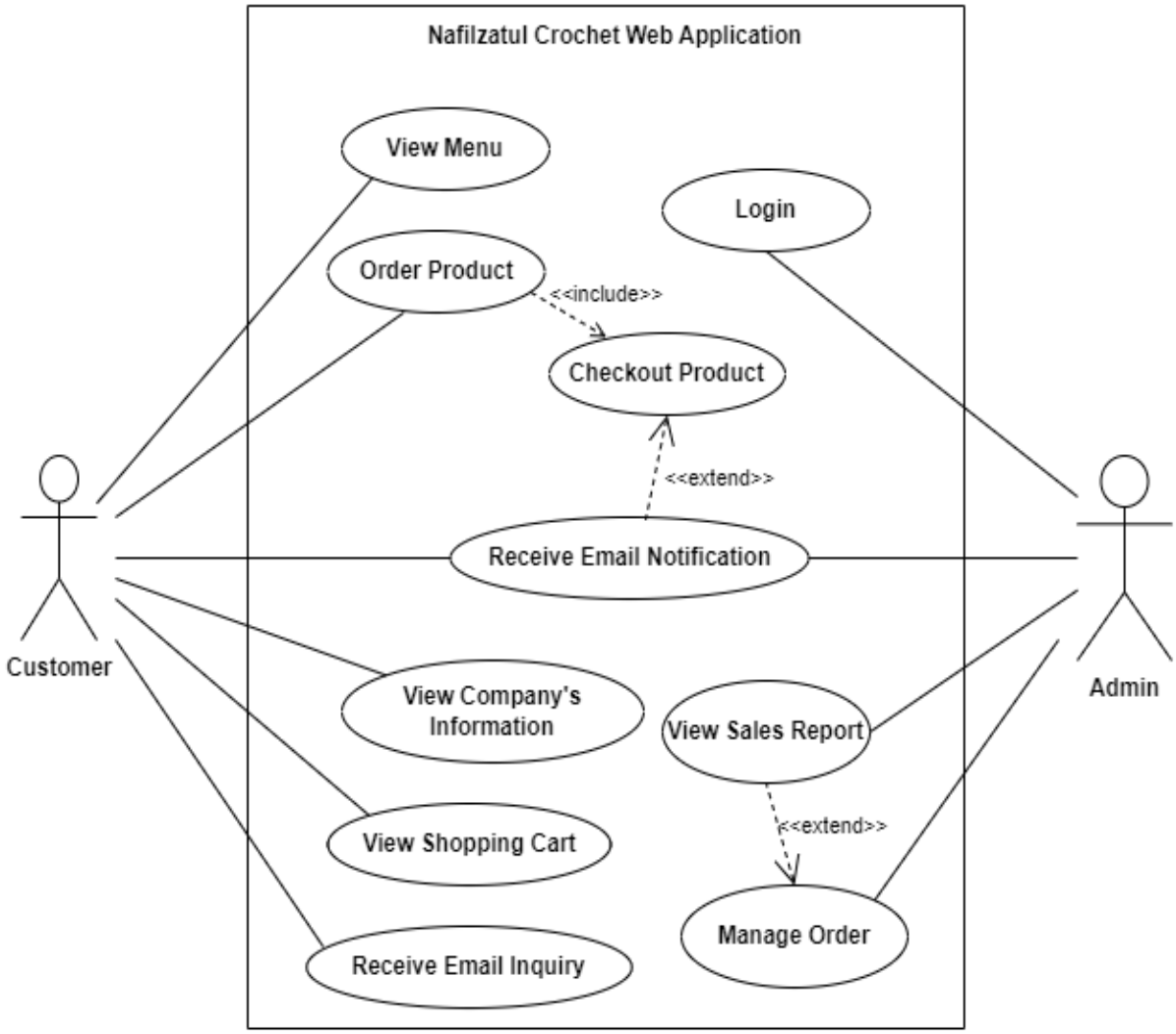
b. Textual user stories

No.	Textual user stories
1	As a patient, I want to register an account so that I can access the counselling web service.
2	As a patient, I want to log in to my existing account so that I can manage my appointments.
3	As a patient, I want to book an appointment so that I can schedule a counselling session.
4	As a patient, I want to receive an appointment notification so that I am reminded of my session.
5	As a patient, I want to view FAQs so that I can get answers to common questions.
6	As a patient, I want to view events so that I can participate in related programs or activities.
7	As a patient, I want to view contact information so that I can reach out for additional support.
8	As a counsellor, I want to update my availability so that patients can book appointments at suitable times.
9	As a counsellor, I want to release a reference form so that I can provide official documentation for patients.
10	As a counsellor, I want to send daily reminders so that patients are informed about their appointments.
11	As an admin, I want to send daily reminders so that patients are informed about their appointments.

Nafizatul Crochet Web Application

An online platform for crochet products where customers browse items, place orders, and checkout, while admins manage sales reports and orders.

a. Use case diagram



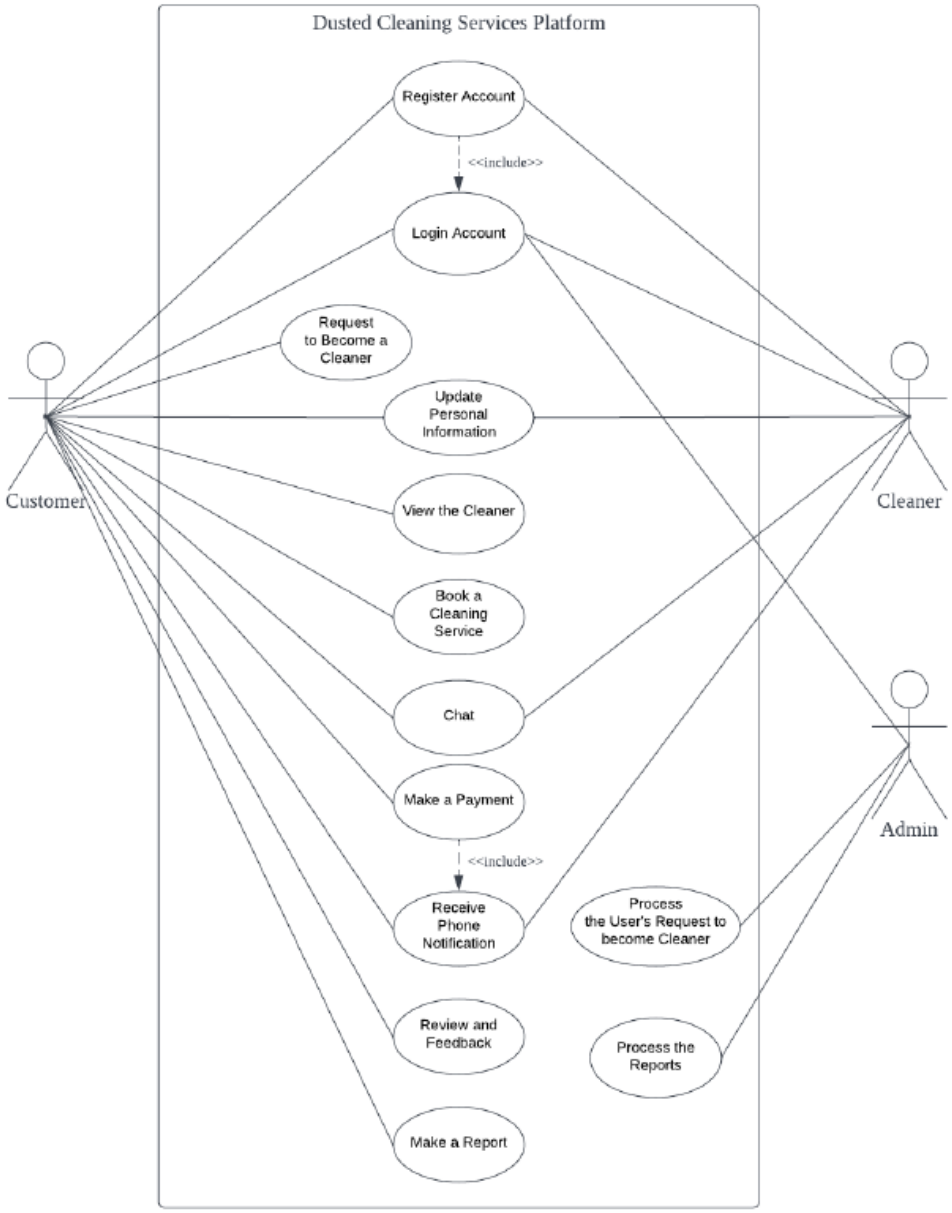
b. Textual user stories

No.	Textual user stories
1	As a customer, I want to view the menu so that I can browse available crochet products.
2	As a customer, I want to order a product so that I can purchase items from the web application.
3	As a customer, I want to checkout products so that I can confirm my order and payment.
4	As a customer, I want to receive email notifications so that I can stay updated on my order status.
5	As a customer, I want to view the company's information so that I can learn more about the business.
6	As a customer, I want to view my shopping cart so that I can review and edit my selected items.
7	As a customer, I want to send email inquiries so that I can ask questions or provide feedback.
8	As an admin, I want to log in so that I can access administrative features.
9	As an admin, I want to view sales reports so that I can analyse business performance.
10	As an admin, I want to manage orders so that I can process and track customer purchases.
11	As an admin, I want to receive email notification so that I can stay up to date on my order status.

Dusted Cleaning Services Platform

A platform connecting customers seeking cleaning services with cleaners. Customers can book services, chat, and make payments, cleaners can request to join the platform, and admins manage cleaner applications and reports

a. Use case diagram



b. Textual user stories

No.	Textual user stories
1	As a customer, I want to register an account so that I can access the cleaning services platform.
2	As a customer, I want to log in to my account so that I can manage my bookings and profile.
3	As a customer, I want to update my personal information so that my details remain accurate.
4	As a customer, I want to view cleaners so that I can select the best option for my needs.
5	As a customer, I want to book a cleaning service so that I can schedule a cleaning appointment.
6	As a customer, I want to chat with the cleaner so that I can communicate specific instructions.
7	As a customer, I want to make a payment so that I can confirm the booking.
8	As a customer, I want to receive phone notifications so that I stay updated about my bookings.
9	As a customer, I want to provide reviews and feedback so that I can rate my experience.
10	As a customer, I want to make a report so that I can flag any issues or concerns.
11	As a cleaner, I want to request to become a cleaner so that I can offer my services on the platform.
12	As a cleaner, I want to chat with customers so that I can discuss their specific needs.
13	As an admin, I want to process requests to become a cleaner so that I can approve or deny applications.

14	As an admin, I want to process reports so that I can address any flagged issues on the platform.