MINAR International Journal of Applied Sciences and Technology
ISSN: 2717-8234
Article type: Research Article

Received: 23/06/2023 **Accepted:** 20/07/2023 **Published:** 01/09/2023

USE CASE TOOLS FOR IDENTIFICATION SOFTWARE REQUIREMENTS: STATE OF ART

Maha Salah Aldin AHMED[1]

University of Mosul, Iraq

Naktal Moaid EDAN[2]

University of MosuL, Iraq

Abstract

The requirement analysis step of the Software Engineering Life Cycle (SELC) is the most crucial in order to ensure excellent quality. Besides, requirements specifications must be compared to the consistency, completeness, and correctness (3C's). Thus, dealing with Natural Language (NL) requirements makes this more challenging than usual., The problems that appear at this stage, if they are not detected or proceed, will lead to errors in the system and sometimes to the failure of the entire system; in addition to the effort required and the cost of maintenance will be large. Accordingly, to overcome these problems, different tools have been developed and implemented to analyse the requirements through Natural Language Processing (NLP). These tools facilitate the process of eliciting requirements through Unified Modelling Language (UML) diagrams. Additionally, creating UML diagrams from NL specifications is a very difficult process. In particular, few efforts have been made in the work of extracting NL specifications, so these diagrams facilitate understanding and analysis of requirements and reduce the time of completion very significantly compared to the manual method. Using automated software engineering tools presents high-quality software that can be produced to help software developers and engineers. The main aim of this research is to concentrate on analysing requirements based on the Use Case Diagram, which is a more popular one in UML diagram. Thus, this research will support other researchers in understanding and specifying the useful tools and mechanisms that can help them to analyse the requirements through Natural Language Processing (NLP) based UML diagrams.

Keywords: *Unified Modelling Language (UML) diagrams; Software Requirement Analysis; Natural Language Processing (NLP); Software Engineering Life Cycle (SELC).*

http://dx.doi.org/10.47832/2717-8234.16.1

[1] maha.21csp15@student.uomosul.edu.iq

[2] naktal.edan@uomosul.edu.iq, https://orcid.org/0000-0003-0799-1858

1. Introduction

1.1. Overview

An essential step that has a significant influence on the software requirements analysis stage of the software development life cycle (SDLC) is automating the collection of requirements. To manage Natural Language (NL) text where the needs are articulated, this automation makes use of cutting-edge Natural Language Processing (NLP) tools and techniques. These criteria can be documented in a variety of ways, such as in unstructured, semi-structured, or structured formats. The semi-structured format is one of them that is frequently used. It should be noted that the use case description contains a variety of keywords that are essential for creating the criteria for extracting UML diagrams.

The division of UML diagrams into structural and behavioral models, each addressing different facets of the software is shown in Figure (1). Class diagrams take up a substantial amount of space among these models since they constitute the foundation of software development. The correct implementation of classes, including their properties, methods, and interactions with other classes, is crucial to the effective creation of software. Automating the extraction of class diagrams is essential in this situation because it enables requirement analysts and stakeholders to work together productively. To automate the selection of class diagram models from semi-structured functional requirements, this work presents a unique set of principles. The use of these guidelines improves the automatic selection of class diagram models' correctness. It should be emphasized though that even with the incorporation of both old and new regulations, there is still opportunity for development. It is difficult to extract class diagrams from semi-structured requirement papers, and even with the existing set of rules, there are restrictions. It is required to investigate new rules and patterns that may handle different phrase forms in order to further improve the procedure. It is acknowledged that rule-based approaches alone fall short of fully addressing the challenge at hand [1].

This survey defines the following as its structure: The relevance of the requirements analysis and the use of UML diagrams in software development are briefly discussed in Section II. Section III includes an overview of recent research. Also, the discussion of this work's in Section IV. Finally, the conclusion is explained in section V.

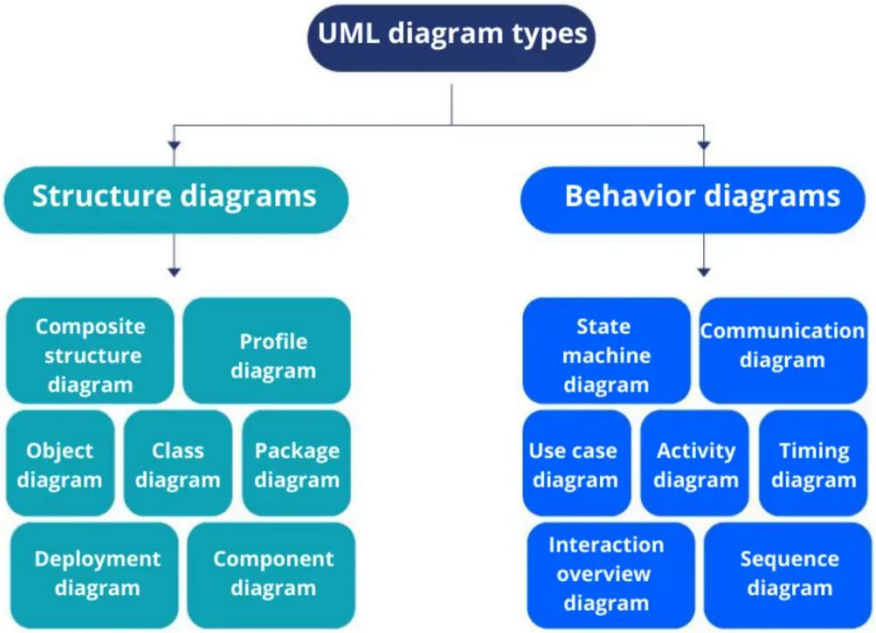

Figure1. Type of UML diagram

Definitions

The semantic processing of human Natural Language (NL) is known as Natural Language Processing (NLP). It includes the processing of text by computers in a way that is understandable to people. NLP focuses on using computer processing to make text comprehensible to people. Software Requirement Specification (SRS) and Software Design Specification (SDS) are two crucial textual outputs in the context of Software Development Life Cycles (SDLCs). NLP approaches may be used to perform several operations on these deliverables. The Object-Oriented Analysis and Design (OOAD) method became the most efficient way of organizing and creating software systems as Object-Oriented Programming (OOP) languages gained popularity [2].

The integration of various modelling standards led to the creation of the Unified Modeling Language (UML). It has emerged as the widely accepted global standard for the development of design documentation, visualization, and construction of software systems. UML provides graphical notations that enable the description and definition of complex systems in a simplified manner [3]. As a foundational element of Object-Oriented Analysis and Design (OOAD), the UML class diagram offers a conceptual approach to address software system design requirements.

NLP systems are used to execute various degrees of analysis on NL requirements based on the information stated above. However, because CASE tools are interdependent, system analysis may be a laborious operation. There have been attempts to use NLP to create UML diagrams and analyze NL requirements. Nevertheless, the use of NLP to assist requirement analysts in NL requirement analysis continues to be a hot matter of debate,

necessitating more improvements and substantial growth. So, any software project utilizing the SDLC must have a strong basis in Requirement Engineering (RE). Besides, it includes the data that the customer has acquired and understood and is used to create software requirements. Also, it is impossible to overestimate the importance of RE in the SDLC given that a poor requirements analysis may cause project delays or even failure [2]. Figure (2), emphasizes the importance of the requirements analysis stage.

Last but not least, UML is one of the most extensive languages used for graphic modelling. Thus, a wide variety of diagrams were taken into consideration, along with a brief explanation of how they were used, the primary benefits and drawbacks of the UML language, and its object-oriented methodology [4].

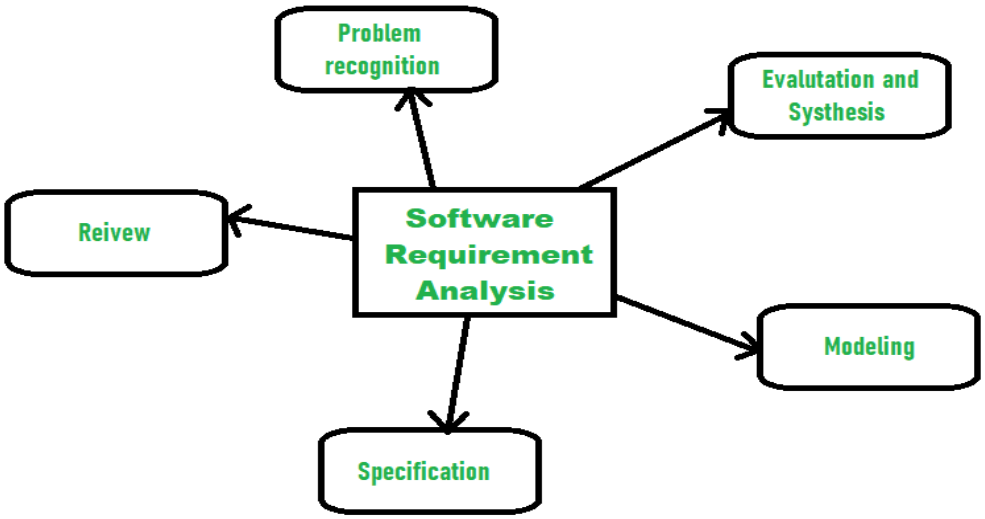

Figure2. Importance of the requirements analysis stage

Requirement analysts in SE spend a massive time processing SRS manually. Therefore, researchers have been attempting to automate this process to make it easier. In addition, the majority of the current methods need some analyst involvement or are challenging to use NLP technologies may be used to effectively tackle challenging and complicated procedures in requirements engineering. But, it might be challenging to translate informal NL requirements into UML diagrams. The approaches that are currently used for this purpose are either limited, computationally expensive or require significant complexity. As a result, researchers have discovered earlier methods that need to be improved upon in order to build UML diagrams. Moreover, they have noticed that prior techniques required developer involvement to produce UML diagrams; also, they have been some recent developments in automated methods, though. These techniques can function without the help of developers since they are made to handle particular input types, including limited natural language or particular text formats. In contrast, previous research

has concentrated on producing informal NL requirements in UML use cases and activity diagrams.

Software analysts can benefit greatly from the availability of a completely automated approach or tool for requirement elicitation and modelling. As a result, it may result in shorter software project development times and lower total costs. Accordingly, numerous tools and approaches have been developed to extract data from NL text and generate UML diagrams. Conversely, NL processing poses several challenges as the following:

1. Ambiguity.
2. Uncertainty.
3. Incompleteness.
4. Illogicality.

These challenges often lead to limitations in the performance of existing techniques [5]. Requirements engineering introduces its own set of difficulties, requiring additional efforts in areas like requirements elicitation, verification, and traceability [6].

2. Related Works

A method to automatically extract UML activity and use case diagrams from Python and Java source codes was introduced by the authors in their article [7]. This application makes the extraction of behavior diagrams for UML easier for programmers in the discipline of Computational Science and Engineering (CSE). This application allows developers to make well-informed judgments about the activities involved in software development and maintenance.

Reverse Engineering (RE) is the focus of this study with UML diagrams serving as the target behavior models. On the other hand, the instrument lacks participant information that depends largely on:

1. Human experience.
2. Especially for crucial use casework.
3. Lacks a graphical user interface (GUI).
4. User reports for evaluation.

Because the tool's results have not yet been verified, these aspects provide difficulties for assessment. it does not address the complexities and voluminous nature of requirements [7].

As mentioned in [8], introducing an effective method for requirement engineering, so describes a process for encoding software requirement descriptions, with a focus on assisting medical operations. The SDLC's requirement engineering stages can be improved with the use of this technique. Therefore, numerous methods are used including the VORD approach as shown in Figure (3). However, it should be highlighted that at the initial step of the analysis, the VORD technique does not explicitly identify objects. The perspective

documentation is converted into an object model for requirement specification at the fourth step, known as system mapping.

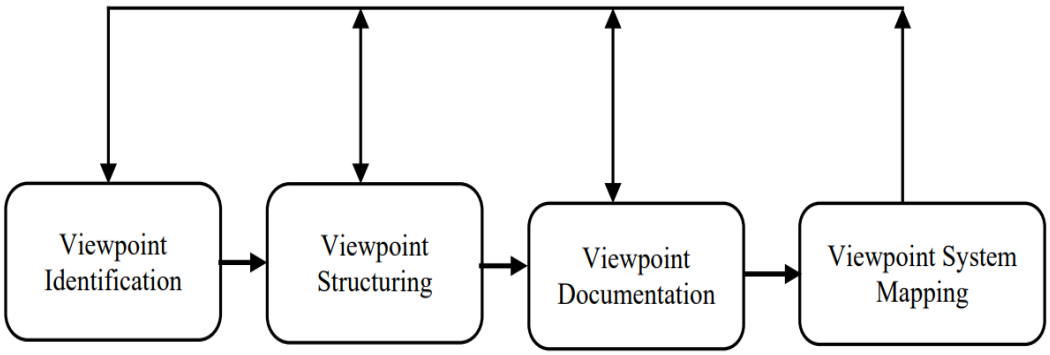

Figure3. word method of Requirement analysis

The authors show how to automate the development of UML diagrams using NL processing techniques [9]. Users of the program can view a document from which the tools for creating the diagrams have retrieved the essential data. The syntax of the activity and sequence diagrams is defined by this method's reliance on the Java programming language. Additionally, it is claimed that commercial software products like IBM Rational Rose and Altova, which offer a drag-and-drop interface for creating UML diagrams without the need for exhaustive requirement specifications can be used. As a result, customers may frequently acquire the same graphic without having to specify their needs in great detail.

In their publication [10], a method and tool were proposed to simplify the requirements analysis process and extract UML diagrams from textual requirements. This approach involved the utilization of NLP and Domain Ontology techniques. Still, the work lacks results, analysis, discussion, and evaluation, indicating a lack of tangible outcomes.

A suggestion for an automated method to derive UML class diagrams from natural language software requirements was made [11]. Using both rules and free-flowing text, this method uses machine learning to accomplish a fully automated generating process. However, this approach's usefulness is constrained when dealing with noun phrases longer than two words, as well as instances of spelling mistakes and vague requirements. While confusing specifications result in the construction of misleading diagrams, spelling errors might result in the creation of superfluous UML classes or associations. This method has an 81% accuracy rate. The estimated top limit of accuracy for the findings is 0.63, although, assuming equal effect from all components, this level of precision is insufficient for practical usage.

In [12], the authors proposed a technique for streamlining the conversion of user requirements into UML class diagrams. The suggested research involves enhancing the production of UML diagrams with NL, enabling more effective and error-free software requirement analysis. When it comes to performing in-depth analyses and comparisons, this technique is constrained. It was neither evaluated using a realistic system nor was it tested in the context of a genuine software development project. As a result, it may not accurately represent real-world circumstances. Additionally, it only concentrates on a particular language, like English. NLP methods applied to requirements text were used to generate UML use case diagrams [13]. The method separated English phrases into their parts, called parts of speech (POS), and gave each one a particular grammatical pattern. Grammar and spelling mistakes were also found using the Ginger Spell Checker (GSC) program. The method has an average recall rate of 69.8% and a precision percentage of 72%. However, several difficulties limited the job. For instance, defining chunking rules was challenging due to the lack of chunking tools for English languages. Additionally, poorly drafted Software Requirement Specification (SRS) documents influenced the rating. The fact that the installation was only carried out on four systems in English.

In [14] concentrated on creating and evaluating a technique and software tool for the automatic production of UML diagrams in the context of object-oriented programming. This approach uses an NLP technique to automatically create UML diagrams from scenario-based requirements that are provided in English using a system called AGUML. It was examined and assessed whether the algorithm and AGUML were accurate. However, more improvements are required to increase the algorithm's adaptability, enabling less constrained scenario texts and supporting alternate flows. Additionally, as any ambiguity or constraints in the input may result in mistakes in the resultant diagrams, the correctness of the created UML diagrams depends on the clarity of the information supplied.

In [15], the researchers presented an approach to generate natural language from formal specifications to aid in communicating complex technical matters to human beings. It used the UML Analysis and Understanding Tool (UMLAUT), an implementation of a UML/OCL to natural language specification tool. However, it has not yet scientifically evaluated this approach. Also, the used algorithm will not work when cryptically names or uncommon abbreviations are used within the model. Including, it handles a comparably small subset of UML/OCL. As well as, if this approach turns out to be valid, future work must incorporate other common diagram types as well as more complex OCL constructs. Additionally, the quality of the generated language needs to be evaluated in detail. The core issue here is to find a reliable metric to grade a natural language specification with or to conduct a larger-scale study as to how the generated specification is perceived. The AGUML algorithm is dependent on the accuracy and completeness of the presented scenarios. Since the algorithm is based on scenario-based user requirements written in natural English, any ambiguities, inconsistencies, or missing information in the scenarios can lead to errors or incomplete UML diagrams.

In [16], a method for doing this was developed, making use of the Model-Driven Architecture (MDA) method. The objective was to automatically convert a series of user stories into UML use case diagrams using NLP approaches, especially using the Tree Tagger parser. This would solve the problem of creating the Computational Independent Model (CIM). Through a case study, the methodology was verified, with validation rates ranging from 87% to 98%. It is important to keep in mind that there are currently only a few ways for creating UML diagrams from requirements at the CIM level, and the suggested approach is unable to handle phrases with several compound nouns, such as "Administrator database manager.". Furthermore, the achievement of generalization and specialization relationships between actors and use cases was not addressed. Also, this research does not evaluate the effectiveness of the proposed process in real-world software development projects, which could limit its practical usefulness. Too, not being able to handle complex sentences or relationships between use cases, may lead to errors or inaccuracies in generated diagrams. The research does not provide a comparative analysis of the proposed technique with other existing methods.

In [17], a method for helping developers create UML models from normalized NL requirements using NLP approaches was described. The approach mainly concentrates on developing an analysis class model (conceptual model) and a use-case diagram. A condensed design class model is then created, which may then be used as a foundation for creating a code model, and a collaboration model is developed for each use case. However, it should be highlighted that this method cannot distinguish between objects and characteristics and suffers from accuracy when choosing objects for bigger systems.

A tool was developed as described in [18], to provide thorough automated support in the development of static and dynamic design models from NL requirements. This strategy is intended for a small subset of requirements, namely those with a word count of under 200 words. Additionally, text normalization is not offered, which may result in information loss when processing bigger requirement papers. While it would seem hard to fully automate this process using NLP, there is yet room for advancement.

The researchers in [19] created a prototype solution to automate the tracing of abstract interactions, minimizing the human work necessary for a group of requirements engineers to identify key use cases. Marama Essential was used in the tool, enabling the integrated visualization of textual requirements as well as graphical representations of crucial interactions and use cases. The use and use of Essential in this situation were examined in the paper.

An alternative user-centric technique known as Use Cases (EUCs) was established to streamline the process of gathering and documenting requirements. This approach is shown in Figure (4). However, due to a lack of proper tool support, EUCs are not frequently used in practice. When attempting to effectively determine the relevant "essential" needs (abstract interactions), requirements engineers face difficulties. Additionally, this extraction technique

was unsuccessfully integrated into the Marama Essential tool for EUC Diagrams, which was created using the Marama meta-tool.

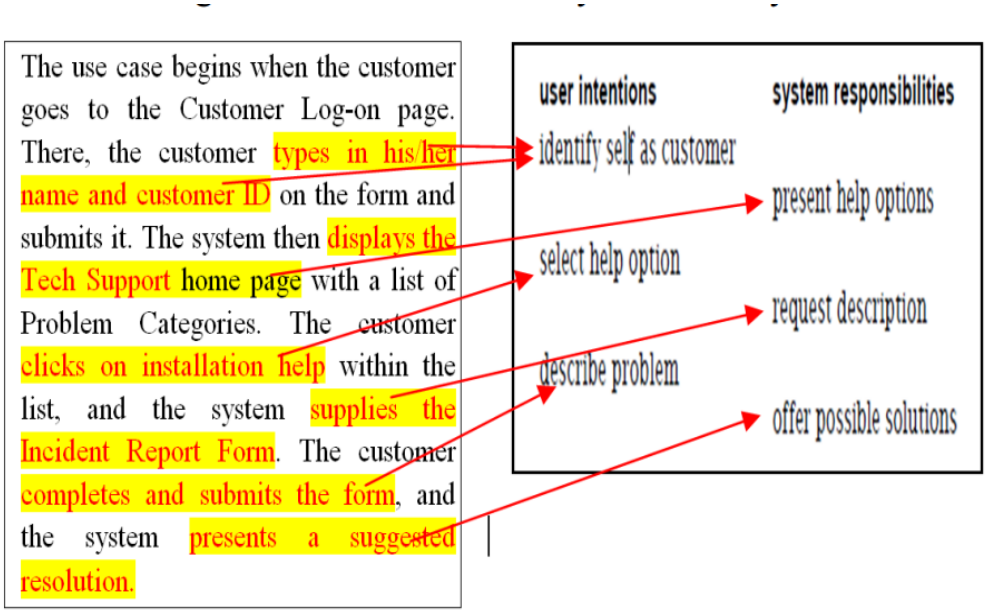

Figure4. Capturing requirements using an Essential Use Case

Despite its limitations, this study supports earlier results addressing the challenges involved in incorporating natural language needs into EUC models. It is crucial to remember that this tool cannot guarantee perfect accuracy owing to linguistic complexity and the frequent occurrence of inaccurate or lacking text requirements. Utilizing round-trip engineering between natural language and EUC model requirements, further assistance is also required in spotting inconsistencies, incompleteness, and redundancies. In addition, the classification in Figure (5) is merely divided into two portions, and no graphics or references to particular methods or classification procedures are included.

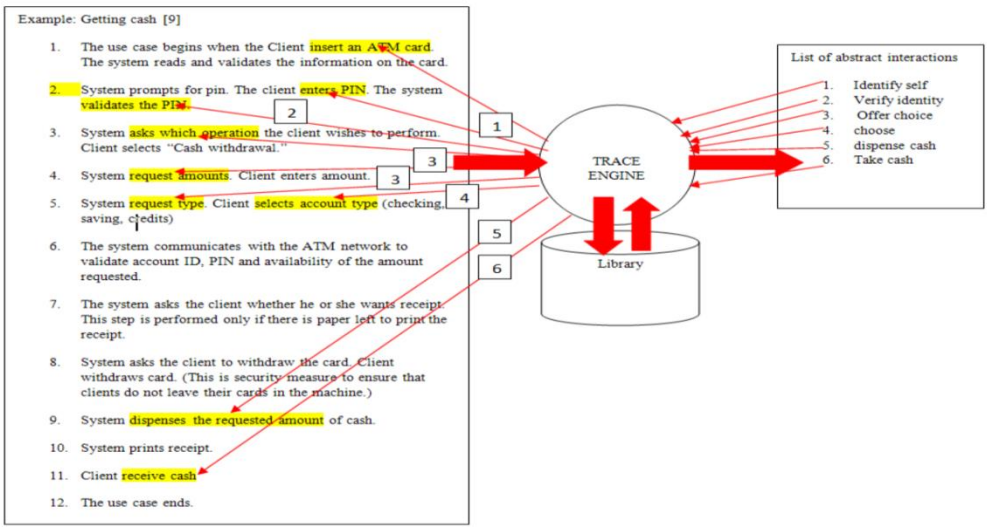

Figure5. Performing an essential interaction extraction to a EUC model and supporting

An analysis of an automated method for finding errors in software requirement specifications was done [20], with an emphasis on finding confusing SRS written in the Malay language. The replies of the participants and Malaysian industry software development were utilized to create the dataset for the study. It is important to note that English has been the primary subject of most of the existing research in this area. The research also emphasizes the difficulties created using the restricted use of boilerplates in requirement formulation, which makes requirement testing and evaluation even more challenging. It is crucial to note that this study serves just as an experimental benchmark. In [20], a thorough examination of an automated method to spot mistakes in software requirement specifications was carried out, with an emphasis on spotting unclear SRS written in the Malay language. The information used in the study was obtained from participant answers and industry practices for software development in Malaysia. It is important to note that the English language has been the focus of the majority of prior studies in this area. The limitations of boilerplate usage in requirement formulation, which make requirement testing and evaluation more difficult, are also highlighted in the study. Recognizing that this study acts as an experimental benchmark to examine the efficiency of the automated procedure is significant.

An automated prototype tool that provides writing capabilities for textual requirements and supports consistency checks across these needs was provided [21]. The ability to validate the consistency of their natural language requirements with other analytical and design representations makes this tool an invaluable resource for requirement engineers and business analysts. With the help of a series of Eclipse IDE plug-ins and the Marama meta-tool set, the tool streamlines the processing of important use cases (EUC) for users, notably requirement engineers. This method considerably decreases the amount of time needed to create EUC models from textual natural language requirements, as shown in Figure (6).

However, due to a lack of tool support, a lack of integration with other modelling methodologies, and a lack of expertise among engineers in extracting crucial interactions from requirements, the use of EUCs has been constrained. The automated tracing tool mentioned in this context also has many shortcomings. It performs as a standalone tool and does not interface with other software engineering or requirements tools, giving users fewer benefits. The application also has some drawbacks, such as poor operation instructions and a lack of a user-friendly visual interface. The tool's database needs to be improved with a wider selection of terms from other fields. Finally, no UML diagram was created that used Marama which has several limitations.

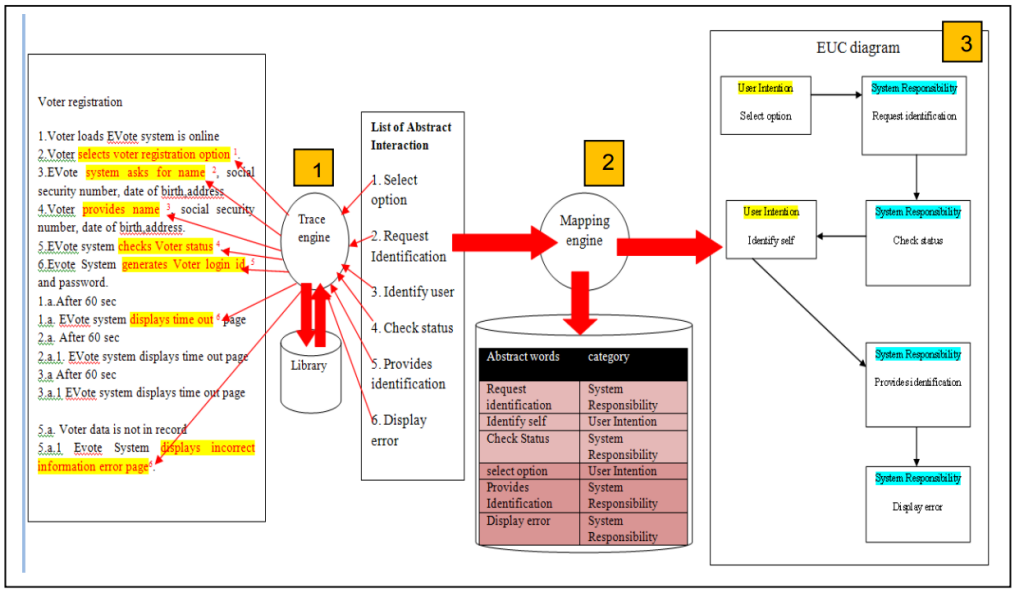

Figure6. Framework for extracting requirement

In [22], the author provided an approach to generating UML use cases and class diagrams from the functional requirement texts using natural language processing and machine learning. The proposed system is capable of providing solutions to generating use cases and class diagrams from the functional requirement text using NLP and ML techniques The Plant UML tool, which makes use of its plain text language and allows for user participation, was used to create the diagrams for this study. The strategy of [23], contrasts with this one in that it focuses on a rule-based strategy to automatically produce UML use case diagrams from functional requirements text or user stories. The prototype created for this study is capable of reading and analyzing functional requirements supplied in texts written in English and automatically producing use case diagrams and class diagrams. It is crucial to highlight that this paper lacks illustrations and in-depth talks that would have improved the technique and outcomes. In essence, it may be viewed as a synthesis or combining of earlier contributions to the subject. In [24], GeNLangUML (Generating Natural Language from UML), a technique the researchers described, intends to produce English specifications from UML class diagrams. An academic case study was used to independently test the Java-based system. For the syntactic analysis of the input names and the sentence-generation verification, WordNet was used in the study. It should be emphasized that these methods' usefulness is restricted to particular domain applications. To put it another way, the researchers found some shortcomings in the other two frameworks that they tried to remedy in the Somerville framework. The user's comprehension of their system needs and preferences as well as the issue area in which the system will be deployed are both taken into consideration in this framework.

The authors developed a prototype solution for their article [25] that helps with the extraction of Essential Use Case (EUC) models from natural language requirements and aids the management of traceability and consistency across these models. The toolkit, which is implemented in Java within the Eclipse IDE, enables collaboration between requirements engineers, end users, and other developers utilizing diagrammatic and textual EUC models. It's vital to note that additional UML diagrams like interaction diagrams, state diagrams, and activity diagrams are not included in this toolbox.

To reduce manual work and produce high-quality use case scenarios (UCS), the authors of the article [26] developed a methodical technique that automates the extraction of different use case parts from textual issue descriptions. Using NLP approaches, the strategy entails creating a template for an intermediate use case and then creating questions depending on the data that was analyzed. POS tags, Type Dependencies (TDs), and semantic roles are extracted from the input text specification using NL parsing and utilized to populate the use case components. The findings show that the information provided is precise, consistent, non-redundant, and thorough, offering a useful support to use case developers for additional analysis and documentation. To enhance the work and its outcomes, this work does not contain any diagrams or discussions. To put it another way, it is only a synthesis of earlier efforts.

In [27] presents a method and supporting tool that allows natural language and semi-formal requirements models to be checked semi-automatically. This tool makes it easier to maintain consistency and analyze correctness and completeness. To assess the precision and thoroughness of the semi-formal representations, the researchers also put forth the idea of key use case interaction patterns. It is crucial to keep in mind that the technology mentioned, Marama AI is specially made to find contradictions in official specifications and might not be appropriate for doing so in other kinds of requirements papers, such as those written in normal language. The technique does not make use of any additional UML diagrams, such as interaction diagrams, state diagrams, or activity diagrams. Marama Meta Tools analysis depends on the quality of the input data, therefore if the data is inaccurate or inadequate, differences might not be noticed. Additionally, just two fields of needs are classified in the technique, potentially leaving out other crucial factors.

Requirement Engineering Analysis & Design (READ), is a system developed by researchers in [28], that creates a UML class diagram using NLP and domain ontology approaches. They suggested a tool that uses the READ tool to demonstrate how NLP and domain ontology approaches may be used to extract UML diagrams from informal NL requirements. The system's accuracy has to be improved, and its ability to produce additional UML diagrams such as use case diagrams needs to be expanded. The extraction of multicity, aggregation and generalization links also needs to be improved. Additionally, it must be possible for editable source code to be generated automatically from the retrieved drawings.

In [29] describes a method for deriving UML diagrams from NL textual requirements and expediting the examination of NL requirements. The method uses heuristic principles and NLP approaches to build use-case and activity diagrams. It was put to use in a case study and tested in an experiment. It should be emphasized that the development of specific use case connections, such as inclusion and extending, is not addressed in the suggested technique. To improve the system's functionality and speed, it is also recommended to include more rules that are particular to information systems domains. The strategy might further benefit from making use of a variety of NL papers and running additional case studies to confirm its efficacy. Its probable failure to reliably capture the precise meaning and purpose communicated in the input text is another shortcoming of the READ implementation of NLP. Furthermore, the program could require a significant quantity of training data to work at its best, which could be a time-consuming and expensive procedure. It is crucial to remember that the accuracy of this method, which is focused particularly on creating class diagrams, depends on how well text analysis works and the methodologies used.

3. Discussions

Following the literature review that was done for this study, earlier efforts mostly focused on creating UML class models from NL requirements. These studies outlined the tools' limits and noted the difficulties still facing this field. They also looked at methods that have been effective in creating UML diagrams and assessing informal NL requirements. However, there was not much focus in these activities on improving the functionality and performance of the systems. They used case studies and NL documents rather sparingly to support their theories. Furthermore, a sizable portion of these papers lacked comments or illustrations intended to enhance techniques and results. Moreover, other research endeavors focused on the utilization of specific tools, which were found to have several limitations. For instance, these tools often operated as stand-alone systems without integration with other requirements or software engineering tools, resulting in limited benefits for users. Furthermore, these tools were often hindered by various restrictions, such as lacking a user-friendly visual interface or unclear aspects that impeded user comprehension of the tool's process and usage. In addition, the databases supporting these tools required enhancement with a broader range of phrases from diverse domains. Notably, no UML diagram was generated using the Marama tool, which also had its own set of limitations. Moving forward, there is a clear intention to develop UML tools capable of addressing most of these issues and supporting analysis across multiple languages. These future tools are envisioned to offer notifications and language corrections to enhance the user's requirements.

4. Conclusion

In this research, the work is concentrate on analysing requirements based on the Use Case Diagram, which is a more popular one in the UML diagram. Thus, it will support other researchers in understanding and specifying the useful tools and mechanisms that can help them to analyse the requirements through Natural Language Processing (NLP) based UML diagrams. Besides, the aim was achieved based on related works, which proved that no UML diagram was generated using the specific tool and has a set of limitations. Moreover, there is a clear intention to develop UML tools capable of addressing most of these issues and supporting analysis across multiple languages. These future tools have envisioned offering notifications and language corrections to enhance the user's requirements. As a result, the UML tools for addressing and analysing multiple languages are needed, as long as is necessary in software engineering.

References

[1] Sanyal, R., & Ghoshal, B. (2018, June). Automatic extraction of structural model from semi-structured software requirement specification. In 2018 IEEE/ACIS 17th International Conference on Computer and Information Science (ICIS) (pp. 543-58). IEEE.

[2] N. Bashir, M. Bilal, M. Liaqat, M. Marjani, N. Malik, and M. Ali, 'Modeling class diagram using NLP in object-oriented designing', in 2021 National Computing Colleges Conference (NCCC), Taif, Saudi Arabia, 2021.

[3] M. Maatuk and A. Abdelnabi, 'Generating uml use case and activity diagrams using NLP techniques and heuristics rules', in International Conference on Data Science, E-learning and Information Systems 2021, 2021, pp. 271–277.

[4] Semenova, V. Tynchenko, S. Chashchina, V. Suetin, and A. Stashkevich, 'Using UML to Describe the Development of Software Products Using an Object Approach', in 2022 IEEE International IOT, Electronics and Mechatronics Conference (IEMTRONICS), IEEE, 2022, pp. 1–4.

[5] S. Ahmed, A. Ahmed, and N. U. Eisty, 'Automatic transformation of natural to unified modeling language: A systematic review', in 2022 IEEE/ACIS 20th International Conference on Software Engineering Research, Management and Applications (SERA), Las Vegas, NV, USA, 2022.

[6] E. A. Abdelnabi, A. M. Maatuk, T. M. Abdelaziz, and S. M. Elakeili, 'Generating UML class diagram using NLP techniques and heuristic rules', in 2020 20th International Conference on Sciences and Techniques of Automatic Control and Computer Engineering (STA), Monastir, Tunisia, 2020.

[7] R. G. Alsarraj, A. M. Altaie, and A. A. Fadhil, 'Designing and implementing a tool to transform source code to UML diagrams', Period. Eng. Nat. Sci. (PEN), vol. 9, no. 2, p. 430, Mar. 2021.

[8] A. Chakraborty, M. K. Baowaly, A. Arefin, and A. N. Bahar, 'The role of requirement engineering in software development life cycle', Journal of emerging trends in computing and information sciences, no. 5, 2012.

[9] S. Gulia and T. Choudhury, 'An efficient automated design to generate UML diagram from Natural Language Specifications', in 2016 6th International Conference - Cloud System and Big Data Engineering (Confluence), Noida, India, 2016.

[10] P. More and R. Phalnikar, 'Generating UML diagrams from natural language specifications', Int. J. Appl. Inf. Syst., vol. 1, no. 8, pp. 19–23, Apr. 2012.

[11] S. Yang and H. Sahraoui, 'Towards automatically extracting UML class diagrams from natural language specifications', arXiv [cs.SE], 25-Oct-2022.

[12] F. Alharbia, S. R. Masadeh, and F. Alshrouf, 'A Framework for the Generation of Class Diagram from Text Requirements using Natural Language Processing', International Journal, no. 1, 2021.

[13] Z. A. Hamza and M. Hammad, 'Generating UML use case models from software requirements using natural language processing, in 2019,8th International Conference on Modeling Simulation and Applied Optimization (ICMSAO

[14] A. Alashqar, 'Automatic generation of UML diagrams from scenario-based user requirements', Jordanian J. Comput. Inf. Technol., no. 0, p. 1, 2021.

[15] M. Ring, J. Stoppe, and R. Drechsler, 'UMLAUT: Synthesis of Natural Language from Constrained UML Models', in Workshop on Design Automation for Understanding Hardware Designs, 2018.

[16] M. Elallaoui, K. Nafil, and R. Touahni, 'Automatic transformation of user stories into UML use case diagrams using NLP techniques', Procedia Comput. Sci., vol. 130, pp. 42–49, 2018.

[17] D. K. Deeptimahanti and R. Sanyal, 'Semi-automatic generation of UML models from natural language requirements', in Proceedings of the 4th India Software Engineering Conference, Thiruvananthapuram Kerala India, 2011.

[18] D. K. Deeptimahanti and M. A. Babar, 'An automated tool for generating UML models from natural language requirements', in 2009 IEEE/ACM International Conference on Automated Software Engineering, Auckland, New Zealand, 2009.

[19] M. Kamalrudin, J. Grundy, and J. Hosking, 'Tool support for essential use cases to better capture software requirements', in Proceedings of the IEEE/ACM international conference on Automated software engineering, Antwerp Belgium, 2010.

[20] M. H. Osman and M. F. Zaharin, 'Ambiguous software requirement specification detection: An automated approach', in Proceedings of the 5th International Workshop on Requirements Engineering and Testing, 2018, pp. 33–40.

[21] M. Kamalrudin, Automated support for consistency management and validation of requirements (Doctoral dissertation). ResearchSpace@ Auckland, 2011.

[22] K. D. Arachchi, AI-Based UML Diagrams Generator (Doctoral dissertation). 2022.

[23] C. R. Narawita and K. Vidanage, 'UML generator-an automated system for model driven development, in 2016

sixteenth International Conference on Advances in ICT for Emerging Regions (ICTer), IEEE, 2016, pp. 250–256

[24] F. Meziane, N. Athanasakis, and S. Ananiadou, 'Generating Natural Language specifications from UML class diagrams', Requir. Eng., vol. 13, no. 1, pp. 1–18, Jan. 2008.

[25] M. Kamalrudin, J. Grundy, and J. Hosking, 'Managing consistency between textual requirements, abstract interactions and essential use cases, in 2010 IEEE 34th Annual Computer Software and Applications Conference, Seoul, Korea (South), 2010.

[26] S. Tiwari, D. Ameta, and A. Banerjee, 'An approach to identify use case scenarios from textual requirements specification', in Proceedings of the 12th Innovations on Software

Engineering Conference (formerly known as India Software Engineering Conference), Pune India, 2019.

[27] M. Kamalrudin, J. Hosking, and J. Grundy, 'Improving requirements quality using essential use case interaction patterns', in Proceedings of the 33rd International Conference on Software Engineering, Waikiki, Honolulu HI USA, 2011.

[28] N. Bashir, M. Bilal, M. Liaqat, M. Marjani, N. Malik, and M. Ali, 'Modeling class diagram using NLP in object-oriented designing', in 2021 National Computing Colleges Conference (NCCC), Taif, Saudi Arabia, 2021.

[29] M. Maatuk and A. Abdelnabi, 'Generating uml use case and activity diagrams using NLP techniques and heuristics rules', in International Conference on Data Science, E-learning and Information Systems 2021, 2021, pp. 271–277.