**Contents**

[1. Introduction. 1](#_Toc132948882)

[2. Related Work. 5](#_Toc132948883)

[3. Proposed Methodology. 14](#_Toc132948884)

[3.1. Use Case Diagram Generation 15](#_Toc132948885)

[3.2. Class Diagram Generation. 17](#_Toc132948886)

[4. Implementation. 21](#_Toc132948887)

[5. Discussion. 25](#_Toc132948888)

[References 27](#_Toc132948889)

**Automatic generation of UML diagrams from user stories using NLP**

# Introduction.

Scrum is an agile methodology [1] that divides a project into iterations (sprints). Each sprint consists of her five phases of running, analyzing, designing, implementing, and testing a series of user stories. User stories are short notes used to describe Agile requirements [2]. In fact, many software vendors use this notation in their Agile software development process [3, 4]. In the context of model-driven architecture (MDA), user stories are described at the level of model-independent computation (CIM). This model represents the highest level of abstraction and describes system requirements in a way that domain experts can understand. MDA [5] facilitates the transformation approach by converting (CIM) to Platform Independent Model (PIM) and (PIM) to Platform Specific Model (PSM) [6]. Much research has addressed the problem of converting (PIM) codes to (PSM) and (PSM) codes. Use case diagrams for level of work (CIM), BPMN, or activity diagrams [7, 8, 9, 10].

On the other hand, requirements analysis is an essential and basic task in software development. Analyzing the quality of requirements extraction and further developing software work products such as various diagrams and formatted text descriptions lead to the success of the project, product or service.

Humans, as analysts, have an innate ability to contextually understand information in text, regardless of the presence of misspelled words, grammatical errors, or indirect references. In the fields of automated natural language processing (TAL) and artificial intelligence, this performance has not yet been achieved and there is still a long way to go. However, humans are limited by several factors, making them error-prone despite the above powers. Therefore, it is not easy to draw conclusions about the working capacity of the machine and the human capacity to work her two fans of the staircase perfectly in a short period of time. Suggests that NLP tools for requirements engineering can be an effective approach to tedious and rigorous work.

Unified Modeling Language (UML) is "a universal visual modeling language for specifying, visualizing, organizing, and documenting the artifacts of software systems," and designs requirements in a more efficient and organized way [11]. It is often the modeling language of choice for designing any part of a system. It is also considered a good way to reduce confusion between requirements analysis and design phases based on diagrams such as class diagrams, sequence diagrams and use case diagrams [12].

Natural Language Processing (NLP), also known as semi-automated processing, is the automatic processing of human natural language. NLP is primarily synthetic and related to linguistics [13]. Requirements are often expressed as sentences in natural language, which can be incomplete and inconsistent. These needs are analyzed using natural language processing (NLP) tools that help parse language and provide automated assistance [14].

There is no fully functional method or tool for generating UML diagrams from informal natural language (NL) requirements. Most existing approaches are highly complex, computationally expensive, or have limitations.

The researcher found that in previous approaches he required developer intervention to generate UML diagrams. Also, relatively recently, there are some fully automated approaches. However, these approaches work for limited inputs such as limited NL or certain forms of text. In previous work he took informal requirements from NL texts and without developer involvement or assistance he generated UML use cases and activity diagrams. Fully automated approaches and tools make requirements extraction and modeling much easier for software analysts. It also saves time and money in the software development process.

Many tools and methods have been proposed and developed to extract information from natural language text and create UML diagrams. However, natural language processing has many problems such as ambiguity, uncertainty, incompleteness and contradiction. Such a lack of NL can undermine the effectiveness of existing approaches.

In addition, large request documents may cause other problems during processing. Additionally, important information may be missing, making the verb completely unintelligible. Software analysts must identify these problems and use their domain knowledge to solve them.

Many tools and techniques have been proposed to deal with this type of problem. However, in practice, these tools are not used in the software development cycle due to many constraints.

Also, most tools require frequent intervention by the analyst to complete the process. Also, most tools are limited to creating class diagrams only.

Our proposal aims to provide a novel solution to the problem of modeling (CIM) by automatically transforming a collection of user stories into UML usage diagrams. To solve this problem, we propose to create a new plugin that automatically converts user stories into UML based on natural language processing (NLP) techniques.

This paper is structured as follows. This section introduces UML and natural language processing (NLP). Section 2 summarizes and compares research work in the field of UML text requirements. Chapter 3 presents the architecture of the proposed method as a reference point for work, describes in detail the steps to convert user stories into UML use case diagrams and class diagrams, and introduces the main algorithms of plug-ins. Section 4 discusses our approach and reports case study evaluations. Section 5 contains some conclusions.

# 2. Related Work.

Although there are few systematic studies related to our research question, we found many peer-reviewed literature reviews discussed in this section. In recent years, many methods of translating open language requirements into UML diagrams have been explored, but few researchers have focused on operational studies and user stories.

Zaho, etc. [45] provides a comprehensive literature review of NLP related to research engineering (RE) and presents all aspects of RE use: identification, mining, classification, modeling, tracking and mapping, and search and acquisition.

They made the NT dimension useful to the software research community.

Omer et al. [46] conducted a study examining the method and effect. Analysis of their strengths and weaknesses is limited by the scores obtained in each study.

Esra et al [44] performed another systematic analysis of NL and UML class counting methods and method maps.

Kumar [15], [16] proposed a novel approach for designing static and dynamic UML models based on language requirements (UMGAR). This method is semi-automatic. In another previous job, he developed a tool called CHESPER (Open Language Requirements Usage Models and Class Score Generator).

Kruchten only develops robust UML models according to the requirements of the Rational Unified Process (RUP) [17].

Using this algorithm, More and Phalnikar [18] generated a UML diagram from the NL specification using the RAPID prototyping tool. It facilitates the research analysis process using natural language processing (NLP) techniques and domain ontologies, and provides methods and tools to extract UML diagrams from required documents. Requirements engineers manually analyze requirements to understand system scope. The time required for analysis and the lack of human analysis justify the need for tools to better understand the system. RAPID (Requirements Analysis for Rapid Diagrams) is a computer tool that helps research analysts and programming students analyze requirements documents, identify key concepts and relationships, and create UML diagrams. The RAPID system is currently being evaluated and will be conducted in two phases: experimental evaluation and scientific evaluation.

Herchi [19] proposed a method for extracting class scores from essential writing, and the method is based on natural language processing (NLP) techniques.

UML class diagrams are the basis for the analysis and design of objects from which other examples derive.

Sometimes designers make changes themselves because of the time and cost of recreating the class image. Therefore, in this case, user requirements may not match the original model, which not only causes major problems in the software maintenance phase, but also affects its use in prototyping and reengineering of new software where the requirements are interrelated.

Automatic natural language processing (NLP) systems for analyzing a given text are based on the following levels: morphological level, lexical level, syntactic level, semantic level, discursive level and pragmatic level [2]. Domain ontology has also been widely used to improve the efficiency of concept identification. Models a specific domain that represents a part of the world. With this type of ontology, organizations, companies or communities describe the concepts of their domain, the relationships between these concepts and of course the instances that are real things that populate its structure.

The main objective of this work is to explore how natural language processing techniques and domain ontology can be used to support the process of object-oriented analysis.

Text input and user queries, the names of the selected classes, their organization and properties are organized in an XML file.

Zhou [20] introduced a method to automatically generate class diagrams from NL requirements documents.

To understand the written requirements, the system uses NLP techniques and applies a domain ontology to improve the efficiency of category recognition.

They use domain ontologies to combine parsers and part-of-speech (POS) tags to refine results and generate candidate classes.

Mich [21] implements an NLP tool called LOLITA (Large Scale Object Oriented Interpreter, Interpreter and Parser).

LOLITA includes pre-processing of user requirements and the integration of all activities into the NL analysis.

The limitation of this proposal is that the tool cannot distinguish between objects, classes, and attributes, and can only retrieve objects from NL.

Vrushali [22] introduced a framework called RACE Requirements Analysis and Class Diagram Building.

Their method is to convert software requirements expressed in NL into class diagrams.

The RACE architecture is very similar to RAPID. In [23], the authors define a semi-automated method for generating analysis class models, design class models, collaboration, and case diagrams based on natural language (NL) requirements.

However, the authors did not describe the transformation in detail. Identification of external classes is done manually. Madanayake [24] conducted a survey to identify the most common modeling techniques in software engineering.

The authors note that entity relationship diagrams, class diagrams, case diagrams, and user stories are the most common.

They also discuss use case compatibility and user stories.

Robeer [25] proposed a method for automatically generating conceptual models from user stories. They implemented a visual storyteller based on a recent NL processing heuristic. To calculate and restore accuracy, the authors ignore attributes and item counts and focus on the details of user stories.

Letsholo [26] introduces a new software tool (TRAM) for automatically generating analytic models based on natural language requirement specifications (NLR). Analysis models are represented by UML class diagrams. To improve the quality of the model, TRAM supports the exchange of information with software analysts or subject matter experts. In addition, the simulation experience is integrated into model building.

Prasad [27] proposed a technique to reduce the complexity of extracting a design model from a design model. This transformation is based on interface representations, data structures, system architecture, and component-level descriptions. Apply concepts and principles at every stage to create high quality software.

Sagar [28] provides a functional requirements (FR) framework for the English language. In order to visualize functional requirements, the authors focus on the automatic search for concepts and their relationships.

Requirements analysts view the conceptual model as an important artifact created during the requirements analysis phase of the software life cycle (SDLC). A conceptual or domain model is a visual model of the target requirements domain. Due to its visual nature, the model serves as a platform for stakeholders to discuss requirements and allows requirements analysts to further refine functional requirements. During the development and execution phases of software, conceptual models can be converted into class diagrams. Even a semi-automated conceptual model can significantly save time in the requirements phase by speeding up the graphical communication and visualization process.

The article presents a diagram for creating a conceptual model from functional specifications written automatically in natural language. Classes and relationships are automatically identified based on functional characteristics. This identification is based on the analysis of the grammatical structures of sentences and the principles of object-oriented design. EER (Extended Entity Relationship) notation is included in class relationships. During post-processing, optimizations are applied to the detected entities and the final conceptual model is presented.

They use optimizations to detect entities in post-processing. Provides a systematic overview of the Yue [29] conversion process.

Model transformation is one of the fundamental principles of model architecture. To create a software system, a sequence of transformations is performed, starting with requirements and ending with implementation. However, the requirements are mostly textual, but not in an easy-to-understand model for computers. Therefore, the automatic transformation of requirements into analytical models is not easy to achieve. The overall objective of this systematic review is to examine the existing literature that transforms textual requirements into analytical models, highlight open questions, and suggest possible directions for future research. The systematic review resulted in the analysis of 20 primary studies (16 approaches) obtained after a carefully designed selection procedure of articles published in journals and conferences and in software engineering textbooks between 1996 and 2008. The conceptual framework aims to provide common concepts and terminology and define a common transformation process.

Karaa [30] defines a new automated tool called Automatic Class Diagram Builder (ABCD) which generates UML class diagrams from natural language NL (user requirements). They used the Stanford NLP toolkit to process NL requirements to extract syntactic and lexical items.

The authors use NLP pattern-matching techniques to derive class diagram elements such as generalization, aggregation, association multiplicity, and composition. A research paper on converting requirements to UML diagrams is presented [31]. This study covers manual work from 1976 to 2015 to automated tools.

The authors present a comparative study and show the strengths and weaknesses of each technique. Discussions will also take place on the combination of engineering and artificial intelligence needs.

Osman [32] summarizes existing tools for inferring data and process models from natural language text. For each proposed tool, they analyzed the associated problem. The resulting data model is checked for the level of transformation automation, completeness, and efficiency. It also offers several case studies in archeology and medicine.

In Mich et al. [33], Mich et al. [34], the researchers propose semi-automatic approaches by developing a tool called LOTIFA which allows extracting objects from user needs without distinguishing between classes, and attributes. However, in Mich et al. [34], the computer scientist intervenes to refine the results obtained. In Zhou et al. [35], the searchers suggest an approach to automate transformation from text requirements to the conceptual model, their goal was to predict the relationship between two concepts and to distinguish betwixt a class and an attribute; indeed, machine learning was used to automate the acquisition of linguistic patterns from training examples.

In Deeptimahanti et al. [36], the authors have developed a tool called UMGAR that generates three models: use cases, conceptual model, and collaboration diagram, indeed, syntactic rules were defined to extract artefacts by using Stanford parser NLP tool.

In Elbendak et al. [37], the authors describe requirements model obtained by a semiautomatic transformation from requirements presented in textual use case diagram, though, this search based on ClassGen tool programmed in Java language, indeed, this tool identifies nouns, verbs, adjectives and adverbs in textual use case diagram to form class diagram. However, a limitation of their methods is oriented to simple sentence structures.

In Herchi et al. [38], the authors developed a platform called DC-Builder that accepts as input a textual data represented by user needs expressing in natural language, then, the platform identifies the class names, their attributes, and their associations in order to classify them in a structured XML file. However, this initial file will be refined using the ontology domain, next, the tool input file is a general requirement document but not user stories, also, the class diagram generated didn't contain the multiplicity.

In Sagar et al. [39], there is no corpus of standard phrases that can be used in needs language, as reference terms represent an unrestricted requirements document. The researchers have developed 38 rules that may not be enough to cover all kinds of sentence structures.

In Thakur et al. [40], the searchers suggest an approach that produces a class diagram from use case specifications, parts of speech tags (POS tags) and typed dependencies (TD), were used to reach their objective. The authors applied TD rules to extract design components such as classes, attributes, relationships, and operations, however, the developed tool analyses simple sentences. The rules used to extract attributes are not valid in most sentence structures, due to the failure of consecutive names processing.

In Lucassen et al. [41], the authors developed a Visual Narrator tool in python; this tool generates a conceptual model output as a Prolog program or OWL ontology 2. However, they generated only the entities and relationships, but not the class diagram. Furthermore, they did not extract the attributes of each entity.

In Elallaoui et al. [42], the researchers used the TreeTagger analyzer (NLP tool) and developed a JAVA plugin to generate the use case diagram from the user stories; the plugin does not handle sentences containing compound nouns. Also; it does not support inclusion or exclusion relationships between use cases.

In Javed et al. [43] study, the conceptual model is generated from an unrestricted format such as general requirements, user stories, or use cases; indeed, new rules are defined for extracting attributes from requirements text. But attribute extraction rule is based on a set previously designed verb. The authors use a set of verbs that indicate attributes; this approach for extracting attributes is not effective. The researchers have not defined rules for generalization.

# 3. Proposed Methodology.

A given user story is expressed at a high level as a statement that represents the customer's needs.

In this section, we present our approach how to automatically convert user stories to UML use case and class diagrams to improve understanding and collaboration between businesses, product owners, developers and testers, significantly reduce time and efficiently implement systems. And our implementation have been done in python using Spacy and PlantUML libraries.

## 3.1. Use Case Diagram Generation

Transforming user stories into use case diagrams has the following key features: The first step is to preprocess a text file containing a series of user stories. This is done using an algorithm that removes all unnecessary words.

The new file then parses the user stories to classify nouns (NN), proper nouns (NP), determiners (DT), and verbs (VV). Apply POS tags to categorize terms as proper nouns, personal nouns, nouns (singular or plural), verbs, and more. Each term in a user story is classified as a part of speech.

Use case diagrams are created using PlantUML.

We can create a model, add elements that can be actors, connections, or use cases, and finally generate the model as a (.uml) file. Actors, associations, and use-case classes describe models for actors, associations, and use-cases. A model class contains a collection of actors, connections, and use cases.

Before creating an actor, we first check whether the actor exists or not. You can describe functional requirements along with use cases. Connections represent the relationship between each actor and use case.

I followed the steps in the pseudo code presented below to extract a UML use case diagram from a user story. The algorithm takes as input user stories stored in text files. After filtering the text file and removing unnecessary terms, I get a new file containing only (noun/compound noun) and (verb).

So, the system takes the new file as input. I have defined two booleans. The first Boolean named (isFirst) detects the first (noun/compound noun) in each user story, and the second Boolean searches the record to see if (noun/compound noun) already exists (line 8). When the first (noun/compound noun) is detected, the loop checks if the word has already been detected in the previous sentence (lines 9-11). If it is foundational, no actors are created (line 12). Otherwise, a new actor is created (line 14). Lines (15-19) allow us to create use cases by finding all (verbs) that follow (nouns/compound nouns) in each user story. Each actor is associated with a use case through a connection relationship. To create these elements, we use PlantUML and the output will generate a (.uml) file containing the full UML use case diagram.

The extraction algorithm is as follows.

|  |
| --- |
| Algorithm 1 |
| 1. Input: {As a customer, …}  2. Boolean: isFirst, isThere  3. store ← actors that already exist  4: currentActor ← the current actor  5: POS ← part of speech  6: isFirst ← True  7: isThere ← False  8: if(((POS==NN)--(POS==NNS)--(POS==NNP)) && isFirst)  9: for each (Actor a)  10: if(isThere ← True)  11: search a  12: currentActor ← store  13: else if (isThere ← False)  14: a ← create Actore  15: if((POS==VB)—(POS==VBP)—(POS==VBN)—  (POS==VBD)—(POS==VBG))  16: verbe ← add(token)  17: if(((POS==NN)—(POS==NNS)) && !isFirst)  18: verbe ← add(token)  19: UseCase u ← PlantUML |

## 3.2. Class Diagram Generation.

To build a class diagram from a user story, you need to extract the components of the diagram: classes, attributes, relationships, and class actions. To do this, we used tokenization, parts of speech, and cross-reference resolution to process user stories. The dependencies between words are then applied to each sentence in the user story. Stemming, which is actually a process of reducing the root of a word, has been applied to avoid class extraction of singular forms such as plural forms, that is, to eliminate class redundancy.

Design element extraction is based on defining rules that use and analyze the type dependencies of each word in a user story. First extract actors and classes, then extract relationships. Attribute extraction is based on compositional relationships between classes. Our approach starts by extracting the classes. Later, when there is a composition between the two classes, and one of the classes does not depend on the other, it will be converted to a property. PlantUML is also used to create class diagrams. This processing was done in the Python language.

Firstly, we determine the verbs, nouns, and adjectives based on Part Of Speech (POS). Moreover, typed dependencies (TD) were used to identify the subject, the object, the compound name (noun-noun) or (adjectivenoun). Table 1, Table 2, and Table 3 describe the rules used to extract several concepts such as actors, classes, relationships, and their classification.

Identification of attributes is based on composition relationships; therefore, classes obtained are refined by removing the attributes, thereafter, the operations will be extracted.

**Table 1. Rules of actor extraction**

|  |  |
| --- | --- |
| Rules | Description |
| AC1 | In user story, the first noun is an actor. |
| AC2 | If the first noun is a proper name, followed by another name, then the second noun is an actor. |

**Table 2. Rules of class extraction.**

|  |  |
| --- | --- |
| Rules | Description |
| C1 | The direct object of the sentence is a class. |
| C2 | The rule C1 is applied but the direct object is part of a compound noun. |
| C3 | The noun that precedes the possessive apostrophe is a class. |
| C4 | The noun after these prepositions (for, of, to) is a class. |

**Table 3. Rules for extracting relationships and their classification.**

|  |  |
| --- | --- |
| Rules | Description |
| R1 | The verb is an association between the direct object of the sentence and the subject: verb (direct object, subject). |
| R2 | Verb with preposition. |
| R3 | Prepositions like “of”, “to”, “for” and “about” detect an association. |
| R4 | Possessive case: “s’”, “my”, “his”… determine an association; Correlation of pronouns with nouns to extract the associations. |

To extract the UML class diagram components, and also the actors, from user stories, we followed the steps described in the algorithm presented below:

|  |
| --- |
| Algorithm 2 |
| 1. Procedure: (Story S, Actors A, Classes C, Rels R, Attributes ATT,  Operations Op)  2. for each s in S  3. p = POS(s)  4: Word\_tokenize(s)  5: Dependency\_parse(s)  6: A = extract\_Actors(s) [AC1, AC2]  7: C = extract\_class(s) [C1, C2, C3, C4]  8: for each s in S  9: R = Extract\_all\_relationships(s) [R1, R2, R3, R4]  10: Comp = filtrate\_composite\_rel(R)  11: for each c in C  12: if (c in comp)  13: ATT = extract\_attribute(Comp)  14: for each r in R  15: if ATT in r  16: Op = r |

This algorithm takes as input an empty set of user stories S and classes C, ATT attributes, relations R and Op operations. It then fills C, ATT, R and Op while applying the extraction rules. The loop on line 2 consists of going through and analyzing each user story. In (lines 3-4), we apply parts of speech and tokenization to classify each word in the user story. The typed dependencies are then detected. The first iteration extracts the actors and classes (lines 6-7). In the second iteration we obtain the relation (line 9). On line 10, the composition relationship is extracted from the relationship list (R) and some classes are converted to properties based on the composition relationship (lines 11-13). Relations that contain attributes are considered operations (lines 14-16).

# 4. Implementation.

In this section, we presented the experimental results of our approach using spacy and PlantUML libraries in python.

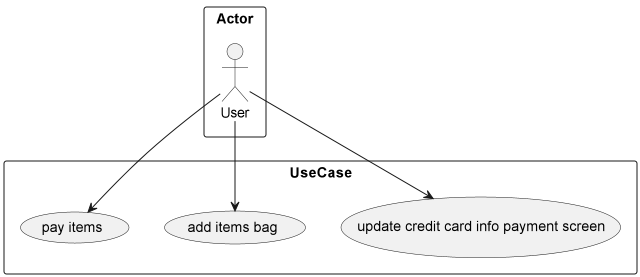
First, we tested how to print use case diagrams with 100 user stories given as follows:

"User should be able to update credit card info on payment screen, add items to bag, and pay for items.", " The chefs need to view orders and recipe instructions for consistent and accurate preparation.", "Managers need to generate reports on sales, inventory, and waste to make informed decisions.", "Overall, the system should aim to optimize operations, reduce waste, and provide an exceptional dining experience for customers." ...

Example 1:

Input user story: “User should be able to update credit card info on payment screen, add items to bag, and pay for items.”

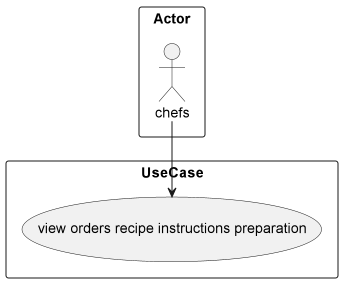
Output is as follows.



Example 2:

Input user story: “The chefs need to view orders and recipe instructions for consistent and accurate preparation.”

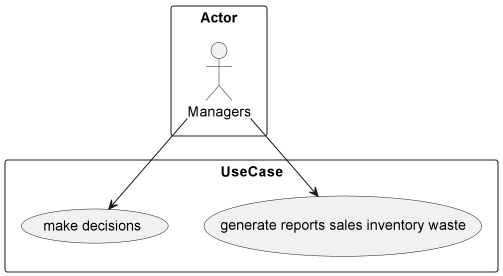
Output is as follows.



Example 3:

Input user story: “Managers need to generate reports on sales, inventory, and waste to make informed decisions.”

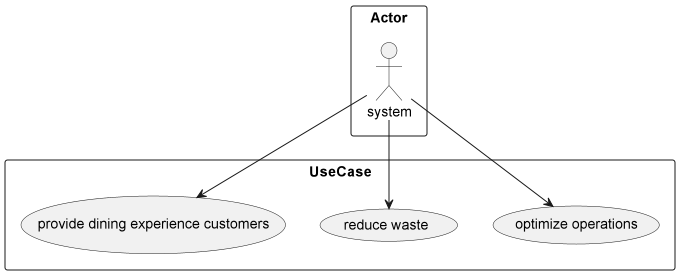
Output is as follows.



Example 4:

Input user story: “Overall, the system should aim to optimize operations, reduce waste, and provide an exceptional dining experience for customers.”

Output is as follows.

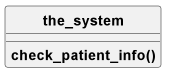


The accuracy was evaluated by comparing what the program generated with what the human thought, and the result reached almost 90%.

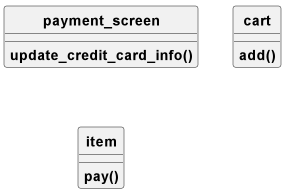
Next, we tested the case of UML class diagram with 100 user stories given as follows:

“Nurse is able to check patient info on the system.”, “Users should be able to update credit card info on payment screen, add items to carts, and pay for items.”, “doctor will update medical records of a patient.”, and so on.

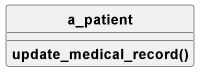
Example 5: For the input “Nurse is able to check patient info on the system.”, the output is as follows:



Example 6: For the input “Users should be able to update credit card info on payment screen, add items to carts, and pay for items.”, the output is as follows:



Example 7: For the input “doctor will update medical records of a patient.”, the output is as follows:



Accuracy of class diagram was evaluated like the case of use case diagram, and 87% was correctly extracted.

# 5. Discussion.

Most of the research in this literature has focused on creating a framework to automate the transformation between user stories and analysis steps to create the required documented use cases and UML class models using of NLP. But few NLP tools can meet the need to create UML models for specific user stories. Rober's research is similar to most recent work, but he did not develop UML models. They automate conceptual models in the form of OWL ontologies based on user stories, while we automatically generate UML use case diagrams and class diagrams based on user stories. The benefits of this technology include simplifying the work of development teams and owners, reducing uncertainty about requirements, and automating design tools. The benefits of creating UML usage diagrams and class diagrams are automatically easier to understand and help designers solidify their user stories, thereby engaging their teams in the design process. It also saves designers time as they can instantly create user story UML models. The Spacy library was chosen because it provides a convenient way to pre-process user stories to include words such as nouns, nouns, pronouns (singular or plural) or verbs. The display of user stories is simple and localized, allowing for categorization. You can easily break down user dialog concepts into individual language blocks. PlandUML is a great tool because you can easily create diagrams in (.uml) format. Our work is very good using this tool. User stories have a simple and understandable structure, which facilitates NLP analysis and sentence generation. The difficulty in developing use case and class diagrams arises from the complexity of the terminology and often from the need to define or exclude relationships between use cases. Our proposed system does not yet support this type of relationship. The challenge of parsing complex sentences to generate case and class attributes will be addressed in future work.

NLP techniques that can be used to analyze a user's speech divide each word of the user's speech into parts of speech. It lets you define requirements, do things in a common language, and produce UML-like results. We believe this technology can make a designer's job easier. This approach helps to better define user stories, shorten class diagrams and use cases for software reviewers, and improve workflow. Use case and class diagrams are useful because they show user needs and functions performed by the application.

# References

1. K. Schwaber, Scrum Development Process. In OOPSLA Business Object Design and Implementation Workshop, Eds. London: Springer, pp. 117-134. (1997) . DOI: 10.1007/978-1-4471-0947-1 11.

2. L. Cao, B. Ramesh, Agile requirements engineering practices: an empirical study, IEEE Software, vol: 25, Issue: 1. (2008) . DOI: 10.1109/MS.2008.1

3. M. Kassab, The changing landscape of requirements engineering practices over the past decade. In: Proceedings of the IEEE international workshop on empirical requirements engineering (EmpiRE). IEEE, pp 18. (2015). DOI: 10.1109/EmpiRE.2015.7431299.

4. X.Wang, L. Zhao, Y. Wang, J. Sun. The role of requirements engineering practices in agile development: an empirical study. In: Proceedings of the Asia Pacific requirements engineering symposium (APRES), CCIS, vol 432, pp.195209. (2014) . DOI: 10.1007/978-3-662-43610-3 15.

5. OMG, MDA. http://www.omg.org/mda/, Accessed January (2016).

6. A. Kriouile, N. Addamssiri and T. Gadi, An MDA Method for Automatic Transformation of Models from CIM to PIM. American Journal of Software Engineering and Applications, 4, pp.1-14. (2015). <https://doi.org/10.11648/j.ajsea.20150401.11>

7. S. Kherraf, E. Lefebvre and W.Suryn,”Transformation from CIM to PIM Using Patterns and Archetypes,” in 19th Australian Conference on Software Engineering, (2008) . DOI:10.1109/ASWEC.2008.4483222.

8. A. Rodrguez, E. Fernndez-Medina and M. Piattini, ”Towards obtaining analysis-level class and use case diagrams from business process models,” Advances in Conceptual ModelingChallenges and Opportunities. Springer Berlin Heidelberg, pp. 103-112, (2008). DOI: 10.1007/978- 3-540-87991-6 15

9. A. Rodrguez, I. G.-R.de Guzmn, E. Fernndez-Medina and M. Piattini, ”Semi-formal transformation of secure business processes into analysis class and use case models: An mda approach,” in Information and Software Technology, 52(9):945 971, (2010). DOI: 10.1016/j.infsof.2010.03.015

10. A. Rodrguez, E. Fernndez-Medina, J. Trujillo and M. Piattini, ”Secure business process model specification through a UML 2.0 activity diagram profile,” Decision Support Systems, vol. 51, no. 3, pp. 446-465, (2011). DOI: 10.1016/j.dss.2011.01.018

11. N.J. Kamarudin, N.F.M. Sani and R.Atan. Automated transformation approach from user requirement to behavior design, Journal of Theoretical and Applied Information Technology, Vol. 81, No. 1, pp. 73-83. (2015). <http://www.jatit.org/volumes/Vol81No1/9Vol81No1.pdf>

12. D. Bell, UML basics: An introduction to the unified modeling language, IBM Developer Works. (2003). <http://www.ibm.com/developerworks/rational/library/769>

13. A. Copestake,Natural Language Processing, Lecture Synopsis. (2004). <https://www.cl.cam.ac.uk/teaching/2002/NatLangProc/revised.pdf>

14. A. Lash, M. Kevin, and M. Gregory. Natural language processing applications in requirements engineering. ASME 2012 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, American Society of Mechanical Engineers. (2012). DOI: 10.1115/DETC2012-71084

15. D.D. Kumar, D.D., M.A. Babar, An Automated Tool for Generating UML Models from Natural Language Requirements. Automated Software Engineering, pp. 680-682. (2009). DOI: 10.1109/ASE.2009.48

16. D.D. Kumar, D.D., R.Sanyal, Static UML Model Generator from Analysis of Requirements (SUGAR). In Advanced software engineering and its applications, (ASEA 2008), pp. 7784. (2008). DOI: 10.1109/ASEA.2008.25

17. P. Kruchten, The Rational Unified Process: An Introduction. Addison Wesley, Second Edition, USA. (2000). ISBN: 0201707101

18. P. More, R. Phalnikar, Generating UML Diagrams from Natural Language Specifications. International Journal of Applied Information Systems, Foundation of Computer Science, vol.1, no.8, pp. 19-23. (2012). DOI: 10.5120/ijais12-450222

19. H. Herchi, W.B. Abdessalem, From user requirements to uml class diagram, CoRR, abs/1211.0713. (2012). <https://arxiv.org/ftp/arxiv/papers/1211/1211.0713.pdf>

20. N. Zhou, X. Zhou, Automatic Acquisition of Linguistic Patterns for Conceptual Modeling. INFO629: Concepts in Artificial Intelligence. (2004). <https://pdfs.semanticscholar.org/b809/d4947a9d544cded70446da3c6bfdf4e0256a.pdf>

21. L. Mich, NL-OOPS: From Natural Language to Object Oriented Requirements Using the Natural Language Processing System LOLITA. Natural Language Engineering, (1996), vol. 2, no. 2, pp. 161-187. DOI: 10.1017/S1351324996001337

22. A. Vrushali, A. Darshana, J. Aarti and L. Dipali, Class Diagram Extraction from Textual Requirements Using NLP Techniques. IOSR Journal of Computer Engineering (IOSR-JCE), volume 17, issue 2, Ver. III, pp. 27-29. (2015). DOI: 10.9790/0661-17232729.

23. D. K. Deeptimahanti, R. Sanyal, Semi-automatic generation of UML models from natural language requirements. In Proceedings of the 4th India Software Engineering Conference. ACM, 165174. (2011). DOI:10.1145/1953355.1953378

24. R. Madanayake, G.K.A. Dias, N.D. Kodikara, User Stories vs UML Use Cases in Modular Transformation, International Journal of Scientific Engineering and Applied Science (IJSEAS) Volume- 3, Issue-1. ISSN (Online): 2409-4285. (2017).

25. M. Robeer, G. Lucassen, J.M.E.M.Van der Werf, F.Dalpiaz, S.Brinkkemper, Automated Extraction of Conceptual Models from User Stories via NLP. In: Proc. of RE (2016). (2016). DOI: 10.1109/RE.2016.40.

26. K. Letsholo, L. Zhao, E.-V. Chioasca, TRAM: A tool for transforming textual requirements into analysis models. in: Proc. ASE 2013, pp.738741, tool demonstration. (2013). DOI:10.1109/ASE.2013.6693146

27. L. Prasad, S. Patidar, Transformation of analysis model to design model. 2010 International Conference on E-business, Management and Economics IPEDR vol.3, IACSIT Press, Hong Kong. (2011). <http://www.ipedr.com/vol3/44-M10006.pdf>

28. V.B.R.Vidya Sagar, S.Abirami, Conceptual modeling of natural language functional requirements. Journal of Systems and Software. 88, 2541. (2014). DOI: 10.1016/j.jss.2013.08.036.

29. T. Yue, L. Briand, Y. Labiche, A systematic review of transformation approaches between user requirements and analysis models. Springer: Requirements Engineering, pp. 7599. (2011). DOI: 10.1007/s00766-010-0111-y.

30. W.B. A. Karaa, Z. B. Azzouz, A. Singh, N. Dey, A. S. Ashour, H. B. Ghazala, Automatic Builder of Class Diagram (ABCD): an Application of UML Generation From Functional Requirements. Journal of Software Practice and Experience, vol. 46, no.12, pp. 14431458. (2016). DOI: 10.1002/spe.2384.

31. M. Abdouli, W.B.A. Karaa, H.B.Ghezala, Survey of works that transform Requirements into UML Diagrams, 2016 IEEE 14th International Conference on Software Engineering Research, Management and Applications (SERA), pp. 117-123. (2016). DOI:10.1109/SERA.2016.7516136.

32. C.C. Osman, P.G. Zalhan, From natural language text to visual models: A survey of issues and approaches. Informatica Economica, vol.20, no.4/2016, 4461. (2016). DOI: 10.12948/issn14531305/20.4.2016.01.

33. Mich, L. “NL-OOPS: From Natural Language to Object Oriented Requirements Using the Natural Language Processing System LOLITA” Natural Language Engineering. (1996) https://doi.org/10.1017/S1351324996001337.

34. Mich, L., and R. Garigliano, “NL-OOPS: A Requirements Analysis Tool Based on Natural Language Processing.”, in Management Information Systems, 2002.

35. Zhou, N, and Xiaohua Zhou, “Automatic Acquisition of Linguistic Patterns for Conceptual Modeling.” INFO 629: Artificial Intelligence, 2002, 1–19. http://daviszhou.net/Research/INFO629Prj.pdf.

36. Deeptimahanti, Deva Kumar, and Muhammad Ali Babar, “An Automated Tool for Generating UML Models from Natural Language Requirements.”, in ASE2009 - 24th IEEE/ACM International Conference on Automated Software Engineering, 2009. https://doi.org/10.1109/ASE.2009.48.

37. Elbendak, Mosa, Paul Vickers, and Nick Rossiter, “Parsed Use Case Descriptions as a Basis for Object-Oriented Class Model Generation.”, 2011, Journal of Systems and Software 84 (7): 1209–23. https://doi.org/10.1016/j.jss.2011.02.025.

38. Herchi, Hatem, and Wahiba Ben Abdessalem, “From User Requirements to UML Class Diagram.” 2013, http://gate.ac.uk/.

39. Vidya Sagar, Vidhu Bhala R., and S. Abirami, “Conceptual Modeling of Natural Language Functional Requirements.” 2014, Journal of Systems and Software 88 (1): 25–41. https://doi.org/10.1016/j.jss.2013.08.036.

40. Thakur, Jitendra Singh, and Atul Gupta, “Automatic Generation of Analysis Class Diagrams from Use Case Specifications.” 2017, http://arxiv.org/abs/1708.01796.

41. Lucassen, Garm, Marcel Robeer, Fabiano Dalpiaz, Jan Martijn E.M. van der Werf, and Sjaak Brinkkemper, “Extracting Conceptual Models from User Stories with Visual Narrator.” 2017, Requirements Engineering 22 (3): 339–58. https://doi.org/10.1007/s00766-017-0270-1.

42. Elallaoui, Meryem, Khalid Nafil, and Raja Touahni , “Automatic Transformation of User Stories into UML Use Case Diagrams Using NLP Techniques.”, 2018, in Procedia Computer Science, 130:42–49. Elsevier B.V. https://doi.org/10.1016/j.procs.2018.04.010.

43. Javed, Muhammad, and Yuqing Lin, “Iterative Process for Generating ER Diagram from Unrestricted Requirements.” ENASE 2018 - Proceedings of the 13th International Conference on Evaluation of Novel Approaches to Software Engineering 2018-March (Enase): 192– 204. <https://doi.org/10.5220/0006778701920204>.

44. E. A. Abdelnabi, A. M. Maatuk, and M. Hagal, “Generating uml class diagram from natural language requirements: A survey of approaches and techniques,” IEEE 1st Int. Maghreb Meeting of the Conf. on Sciences and Techniques of Automatic Control and Computer Engineering MISTA, 2021.

45. L. Zhao, W. Alhoshan, A. Ferrari, K. J. Letsholo, M. A. Ajagbe, E.-V. Chioasca, and R. T. Batista-Navarro, “Natural language processing for requirements engineering: A systematic mapping study,” ACM Computing Surveys, Vol.54, No.3, Art. 55., 2021.

46. O. S. Dawood and A.-E.-K. Sahraoui, “From requirements engineering to uml using natural language processing –survey study,” EJERS, European Jour. of Engg. Research and Science Vol. 2, No. 1, 2017.