Requirements and Design Consistency: A Bi-directional Traceability and Natural Language Processing Assisted Approach

Omer Salih Dawood Omer, Abd-El-Kader Sahraoui, Mukhtar M. E. Mahmoud,
and Abd-El-Aziz Babiker

Abstract — Requirements Engineering (RE) is a crucial and essential phase in the Software Development Life Cycle (SDLC). The design phase follows it. Traceability is one of the core concepts in software engineering; it is used to follow updates to make consistent items. This paper aimed to cover consistency through bi-directional traceability between requirements and design phase in a semi-automatic way. The Natural Language Processing (NLP) was used to analyze the requirements text and generate a class diagram; then, the generated items can be traced back to requirements. We developed a novel process to support consistency and bi-directional traceability. To ensure our proposed process's practical applicability, we implemented a tool named as Requirements and Design Bi-directional Traceability (RDBT). RDBT receives textual format requirements, performs NLP tasks (Tokenization, Part-of-Speech Tagging, etc.), generates UML class diagram, and finally performs traceability management to ensure consistency of requirements and UML class diagram. The work evaluation reveals good results, which indicates it can be used efficiently as a guide to generate the UML class diagram semi-automatically and manage traceability.

Index Terms — Natural Language Processing, Software Design, Requirements Traceability.

I. INTRODUCTION

Completing successful software projects within its approved plan is a critical challenge. As in [1], around 69% did not deliver due to improper requirements management, lack of suitable user input, inadequate staff training, incomplete requirements, rapid change, ambiguity, and poor contract management. Obviously, poor requirement management has a lion share in the failure of software projects.

There are many phases for software development; each phase depends on the previous ones. Requirements' engineering is the first phase. In this phase, the requirements are gathered, analyzed, validated, and documented. The final output is a Software Requirements Specification (SRS). Commonly, requirements in SRS are written in Natural Language (NL).

Requirements engineers can write requirements in their sentences. During system analysis and design, it is crucial to ensure that the needed design phase requirements are covered to make a consistent design to make high-quality software.

NL's use as requirement descriptions causes several issues, such as ambiguous, incomplete, and inconsistent. Hence, the requirements engineer's primary duty is detecting, dealing, and fixing these issues [2]. Requirements' specifications are the formal description of business processes; as a life fact, business processes have dynamic nature, i.e., business processes comprise continuously changing. Therefore, requirements traceability is crucial to establish and maintain consistency between heterogeneous modes through the system development lifecycle [27].

On the other hand, NLP applications to requirements can be used to verify the SRS document's completeness and consistency [7]. Building and traceability and consistency are essential to achieve consistency between models. Consistent requirements should be linked to other requirements [22]. Based on these, we need to cover the area of requirements consistency between SRS and SDD through a novel methodology. This approach can be treated with an NL text and UML class diagram, and a consistent and traceability matrix was built. So, the traceability matrix can be used to ensure consistency [27]. There are the types of inconsistency in UML model redundancy, Conformance to Constraints and Standards, and change [14]. In our paper, we considered change inconsistency.

The automation allows for consistency, and errors can be avoided in earlier stages. One method to build traceability between requirements and model use Natural language Processing (NLP) techniques to process requirements text to obtain UML diagrams using different methodologies [2].

This work is an extension of our previous study [34], which covers the auto-generation of UML diagrams from requirements without going in-depth for inconsistency types. A framework enhanced to minimize the auto-generation of UML diagrams applies consistency between requirements and design.

Submitted on February 14, 2021.

Published on April 26, 2021.

Omer Salih Dawood Omer, Department of Computer Science, College of Arts and Science Wadi Aldawasir, Prince Sattam Bin Abdulaziz University, Saudi Arabia, Sudan University of Science and Technology, Sudan.
(e-mail: o.dawood@psau.edu.sa)

Abd-El-Kader Sahraoui, LAAS-CNRS, Université de Toulouse, CNRS, UT2J, Toulouse, France.
(e-mail: sahraoui@laas.fr)

Mukhtar M. E. Mahmoud, Faculty of Computer Science and Information Technology, University of Kassala, Kassala, Sudan.

(e-mail: mukhtaredris@gmail.com)

Abd-El-Aziz Babiker, Sudan University of Science and Technology, Khartoum, Sudan, and Majmaah University, College of Sciences and Humanities Ghat, Majmaah, KSA.

Fig. 1. Traceability between requirements and design [4].

The paper covers moving from SRS to Software Architecture Design Description (SADD) in a semi-automatically to make bi-directional traceability in a consistent matter to ensure requirements consistent with the design. In the literature review section, lots of research were reviewed; these researches cover generation design diagrams from requirements without considering consistency and traceability [7]-[13], [15]. There is some research cover traceability but did not deal with consistency management [16]-[25]. In contrast, others deal with consistency but did not deal with auto-generation UML and requirements processing and how traceability can be managed [14], [29]-[31]. We developed a methodology to make consistent requirements and UML class diagrams through auto-generation class diagrams and bi-directional traceability. As in Fig. 1, the research covers the first two stages (SRS) and (SADD). Both forward traceability SRS to SADD, and backward traceability SADD to SRS.

The rest of the paper is organized as follows. Section 2 investigates the background and related work, while section 3 details the methodology. Section 4 covers our case study, while in section 5, we evaluate our proposed methodology. Finally, section 6 concludes the work and gives insights regarding future work.

II. BACKGROUND AND RELATED WORK

Requirements Traceability and Natural Language Processing (NLP) are the two main terms of this research, where they have been defined in this section under the preliminary's subsection. In the rest of this section, the related works have been analyzed by highlighting their strengths and weaknesses.

A. Preliminaries

Requirements Traceability: Based on international standards ISO/IEC/IEEE 29148 [3] requirements traceability defined as identification and documentation of the derivation path (upward) and allocation/flow-down path(downward) of requirements in the requirements set. It should be established to maintain the SRS document and trace it through the lifecycle of the system. Requirements Traceability is frequently used as a single point of accountability for tracing a requirement back to the source of the requirement and forward through the life cycle to assess that the requirement has been met [3]. Tracing the requirement specification during software development in all SDLC phases is vitally important [17]. According to [18], traceability management

has been established to cope with the dramatically changing requirements due to add, delete, or modify the requirement features. Traceability management is concerned with ensuring the consistency between requirements documents and design face. For instance, between SRS and class diagram [19]. Involving traceability management in the software development process demonstrate that requirements have been satisfied [20]. Bi-directional traceability provides a means of documenting and reviewing the relationships between layers of requirements that capture certain aspects of the design [3]. In order to maintain and manage traceability in a project, it is common to set up something called traceability matrixes [18].

Natural Languages Processing (NLP): Natural language processing can be defined as processing human language in an automatic (or semi-automatic) way [5]. NLP's goal is to identify the computational machinery needed for an agent to exhibit various linguistic behavior forms, i.e., Scientific Goal. It also designs, implements, and tests systems that process natural languages for practical applications, i.e., engineering goals [6].

B. Analysis of Related Works

The related works have been classified based on work output in two groups: the first group includes research that generates UML diagrams from SRS. The second group includes researches that suggested a mechanism for requirements traceability.

1. From Requirements to UML Transformation

In related works, several studies are dealing with the generation of UML diagrams from requirement specifications. Although these studies have the final aim of facilitating UML generation from requirements, they are still varying in methodologies or the final outputs. Some produce use cases, while others produce class, sequence, package, or even activity diagrams in the following, the works that have handled UML transformation requirements.

The work in [7] suggested managing textual requirements through application-specific ontologies and NLP. The study [8] proposed a method of automated the process of generating a package diagram from requirements to support the design process. In [9], [11], the Activity diagram and sequence diagram were generated, wherein [10] use case diagram was generated using heuristic rules. The work in [12] proposed a tool to generate a class diagram from requirements, then generate code from class. Chamitha et al. [13] proposed a workflow for generating use cases and class diagrams. They used web browsers to enter the text, finally process it using NL processing techniques, and then generate candidate diagram elements. Keletso [15] developed a tool for constructing an analysis model from the natural language specification of requirements (NLR) automatically. Imane et al. [16] developed an approach to generate a use case diagram from Semantic Business Vocabulary and Business Rules (SBVR). SBVR is requirement representations in a structured language instead of using natural language. In [26], they used analysis based approach to generate a class diagram from natural language requirements through dependency. However, their model neglected, providing an inconsistency between requirements and class diagrams. [32] proposed a tool named

CM-Builder to support the auto transformation from requirements to UML diagrams. However, a detailed description of consistency between requirements and generated diagrams are missed. The studies mentioned above have concentrated on the generation of UML diagrams without considering the bi-directional traceability to follow changes in both sides. Table I shows a summary of the previous discussion by highlighting the goal and limitation of each work.

2. Requirements Traceability

In the following, the works that have dealt with requirement traceability management have been explored and discussed.

In [18], the framework for modeling requirements, traceability, and UML patterns was introduced as a communication tool between requirements and other software artifacts. The work in [19], proposed a formal methodology that utilized context-free grammar to denote use case, activity, and class diagrams. In [19], traceability rules were developed to ensure the use case mapped to activity and class diagrams. However, this is uni-directional tractability as their methodology lacks opposite traceability, i.e., traceability from (SDD) to SRS. In [21], guidelines for supporting traceability were introduced with manual generating of the UML diagram. In [22], a model labeled as Requirement Interchange Format (ReqIF) was proposed to exchange requirements traceability information between different Requirement Management Tools (RMT) and designing tools that are limited to XML only. The qualitative data analysis to improve the traceability of requirements was used in [23] to focus only on elicitation

methods. In [24], the meta-model was proposed to provide explicit linking with requirements to enables traceability. However, the description of the system designed has been ignored in [24]. In [1] proposed a framework for requirements management that support requirements traceability. It consists of planning, execution, and management phases. The work in [1] disregards the design details.

In [14], three types of UML model's inconsistency have been introduced based on rule based. The rules are to detect inconsistencies between UML diagrams. However, inconsistencies between the UML diagram and SRS do not detect. In [28], formal methods for detecting five types of consistencies in the Use Case diagram has been presented. Work [28] has neglected to check for inconsistency between the Use Case diagram and SRS text. Work [29] provides an intuitive approach to analyze UML diagrams' consistency using Web Ontology reasoners. The consistency analysis is exclusively implemented only on UML diagrams. In [30], integration between UML and Service Refinement (SR) has been introduced to ensure consistency. However, consistency between SRS and UML diagrams has not been considered. Table II summarizes the most relevant studies that deal with requirement tractability.

In summary, from the works mentioned above, we can summarize the most crucial research gap in the lack of a model that can simultaneously perform Auto-generation of UML diagrams, provide bi-directional requirement tractability between SRS documents, and UML, and produce a tractability matrix.

TABLE I: SUMMARY OF STUDIES THAT DEAL WITH REQUIREMENT TO UML TRANSFORMATION

#	Generated Diagram	Goal	Limitation
[7]	No diagram generated	Scan and tag a set of requirements and allow system engineers to maintain a comprehensive, valid, and accurate body of requirements	It is missing a clear description of consistency and traceability
[8]	Package diagram	To have a systematic solution that recommends design choices based on system requirements	Do not cover traceability and consistency
[9]	Activity and Sequence Diagrams	To minimize the errors that arise in a traditional system	Do not cover traceability and consistency
[10]	Use Case Diagram	Facilitate the requirements analysis process	Do not cover traceability and consistency
[11]	Activity and Sequence Diagrams	Manual transformation from requirements to UML	Do not cover traceability and consistency. It also missing automation
[12]	Class and Sequence Diagrams	Supporting the Analysis stage of software development	Do not cover traceability and consistency
[13]	Use Case and Class Diagrams	To build quick, accurate, and intelligent software for generating UML	Do not cover traceability and consistency
[15]	No Diagram generation	Facilitate the transformation of a Natural Language Specification Requirements to its target software mode	It is missing a clear description of consistency and traceability
[16]	Use Case Diagram	Generating Use Case Diagram from Semantic Business Vocabulary and Business Rules (SBVR)	Discussed traceability between (UCD) and (SBVR) without covering consistency
[26]	Class Diagram	Generate class diagram automatically from Natural Language requirements through a dependency analysis based approach	Do not cover traceability and consistency
[32]	Class Diagram	Generate class diagram automatically from Natural Language requirements	Do not cover traceability

TABLE II: SUMMARY OF STUDIES THAT DEAL WITH REQUIREMENT TRACTABILITY

Study	Goal	Limitation
[17]	Proposed architecture for traceability management systems to the product life cycle.	Traceability covered in general through product life cycle without considering Requirements Traceability Matrix.
[18]	Build a framework suitable for modeling traceability patterns allow communicating best practices to other software projects.	Traceability is done between requirements without covering consistency between requirements and design.
[19]	Ensuring traceability and verify consistency among three phases use case, activity & class diagrams.	They did not perform any processing to the requirements, and no traceability matrix was defined.
[20]	Apply traceability information models in practice.	They did not perform any processing to the requirements and did not consider consistency issues.
[21]	Address the issue of traceability mapping between requirements and UML that may cause the problem of inconsistency. It provides traceability guidelines includes meta-model and process step	They did not perform any processing to the requirements. The UML diagram is built manually.
[22]	Use of Standardized ReqIF Format to interchange traceability information between requirements tools and UML tools.	Did not generate a UML diagram, no consistency information, and no traceability matrix defined.
[23]	Use Qualitative Data Analysis to enhance requirements traceability.	It only concentrated on elicitation methods and did not take consistency into account.
[24]	Provide an explicit bridging between the different requirements and design decisions.	It did not offer details about UML diagrams that represent the design.
[25]	They built a framework to support requirements management and traceability management.	The framework is general and did not show the design in detail.
[14]	Proposed Rule-based to detect and trace inconsistency in UML modes. They introduced three types of inconsistency.	Cover inconsistency in UML models without considering consistency with SRS.
[28]	How to check the consistency of requirements which are modeled by UML use cases and constraints.	No requirement text processing.
[29]	Make use of Web Ontology Language to automatic analysis and handling of inconsistency between three of UML diagrams.	They did not cover consistency between Requirements and UML diagrams.
[30]	Utilize a synthetic approach to support the formal requirements modeling and analysis as well as formal verification.	They covered some UML diagrams without making the process of handling and processing.

III. METHODOLOGY

In this section, the methodology for developing the proposed model has been explained and discussed in detail. The methodology consists of two main phases, as in figure 3. The first phase is a requirement processing where the output is a set of tagged texts. It consists of the following steps: rewritten requirement text in a specific way, filter and remove the stop words, perform text tokenization, and apply POS tag. The output of this phase is prepared SRS. The second phase is the auto-generating of a Class diagram from the prepared SRS. In the following, the detailed descriptions of the two phases have been provided.

A. Phase 1: Requirements Processing

The following steps are used for preparing requirements.

1. Writing requirements in specific rules/patterns:

In this step, SRS has been revised and rewritten by classifying requirements into related groups. Each group should be documented in a single form. Redundancies and unnecessary information should be removed.

2. Removing stop-words

Stop word is defined as a word with less significance, such as the, at, which, on, etc. Removing stop words will facilitate text processing. In case of need to some of these words, programmatically can be ignored. For example, we should keep (is a) between classes in the generalization relationship and not remove it.

3. Requirements text tokenization (lexical analysis)

After removing stop words, requirements have been divided into tokens as it is easier to process a single word than a set of words. To implement tokenization, we have used OPEN NLP Tokenizer.

4. Part-of-Speech (POS) Tagging

The final process in phase one is a POS tagging, which forms a word in a corpus to an equivalent part of a speech tag,

based on its context and definition [25]. The POS tagging applied to tokens to markup noun, verb, etc. OPEN NLP POS Tagger has been used to generate the POS tagging. Then we can apply a pattern using a regular expression to extract the candidate class elements. Table III summarizes phase 1, where the input, output, and used tool has been defined.

TABLE III: SUMMARY OF PHASE 1

Step	Input	Output	Tool
Writing requirements in a specific rules	The System SRS	Classified SRS	Manually
Removing stop-words	Classified SRS	SRS without stop words	Manually
Requirements text tokenization	SRS without stop words	SRS as tokens	OPEN NLP Tokenizer
Part-of-Speech (POS) Tagging	SRS as tokens	Corpus	OPEN NLP POS

B. Phase 2: Class Diagram Generation, Traceability, and consistency Management

The following steps are illustrating phase 2.

1. Rules Management

We proposed to save predefined rules in the database when chunking is needed for the pattern retrieved from the database and chunk the text using a regular expression. The domain experts will define rules that can be used to generate UML elements. Here are some examples of used patterns [34]:

Class identification rules/Patterns

- Proper noun, singular (NNP)+ Noun, singular (NN)+ Proper noun, singular (NNP)

- Proper noun, singular (NNP)+ Verb, past tense

Method identification rules/Patterns

- Verb, gerund, or present participle + Proper noun, singular (NNP)

- Noun, singular (NN) + Noun, singular (NN) + Proper noun, singular (NNP)

- c. Preposition (IN) + adjective (JJ) + Noun, singular (NN)
Attributes/Patterns
a. adjective (JJ) + Noun, singular (NN)
Relationship rules/Patterns
a. Verb, gerund or present participle (VBP) + Proper noun, singular (NNP)
b. Verb, gerund or present participle (VBP) + Verb, gerund or present participle (VBG)

2. Rule Engine and UML Database

This module work as a database to store patterns and UML class diagram elements. All classes, attributes, methods, and relationships are saved in this database. The databases are central and save all traceability information and used later to generate reports.

3. Apply Rules/Patterns on Requirements (Chunking)

In this phase, rules were retrieved from the database and applied to POS tagged text using a regular expression to obtain specific elements. In our case, to obtain candidate classes, attributes, operations or methods, and relationships.

4. Select Class Elements

After requirements are chunking, the candidate elements can be selected and saved in the database. The class diagram may contain classes, attributes, methods, and relationships with other classes. The SQL databases can be exported and imported into many models.

5. Class Diagram Generation

In this phase, the diagram is displayed based on elements selected in the previous stage. The class diagram is generated in semi-automatic ways and can be managed, edited, or removed from diagrams. We obtained a fully matched diagram with all class elements, attributes, classes, methods, and relationships generated and matched with manually generated diagrams.

6. Edit Requirements / Design

We may need to trace changes in requirements or class diagram elements. When we change in requirements, this change must be tracked and reflected in the design, and vice versa. When design changed, the change must be traced and reflected requirements.

In this phase, if any change happens to requirements without reflected in the class diagram or any change in the class diagram not reflected in requirements, this causes inconsistency. We manage change inconsistency through bi-directional traceability to ensure both phases are consistent. All information saved in the database, which update, source of change, and inconsistency.

7. Display Inconsistency in Requirements and Design Reports

An inconsistency report can be generated in case of inconsistency between requirements and design (updates not traced and reflected from phase to phase). This report helps us to know what change happened and applied well in both requirement and class diagram, so we can handle and manage inconsistency between these phases.

8. Generate Traceability Matrix

This is a crucial phase; the traceability matrix links both requirements with design elements. It shows all the need information that we need for traceability and inconsistency. It is one of the final outputs of our methodology. Its shows requirements Id, Traceability Id, source of change.

Fig. 2. General overview of methodology.

As in Fig. 2, the SRS document's requirements transformed into a UML class diagram semi-automatically; this will speed up the design generation and decrease error expectation. The Traceability matrix handles both design and SRS. When any changes happen, it can be traced from source to destination. By doing so, we can ensure the achievement of consistency between requirements and design stages.

IV. CASE STUDY

In this section, the proposed model has illustrated by using a case study. Eclipse as IDE, Java as a programming language, and *Apache OpenNLP* have been selected as toolkits to process natural language text.

The case study has been borrowed from [31] by applying some changes to sentence syntax. As has been mentioned in phase 1, all requirements were written in a single form. The critical change that is applied to requirement text is to rewrite requirements based on our proposed rules. Each sentence should be written in a specific format. In this section, we also describe the implementation of the case study.

Fig. 4 shows the main form used to enter SRS. This SRS should follow defined rules, each statement written in a specific way. Fig. 4-19 represent snapshots of our proposed tool.

Stop words have been removed. Fig. 5 shows SRS after removing stop word.

The next step is applying the POS Tag. Fig. 6 shows SRS after applying POS Tag. By preparing SRS as in Fig. 6, the process reaches the end of phase 1. Hence, phase 2 should start by apply rules on tagged text to extract candidate class elements. Regular expressions were used to extract the needed elements for applying these rules.

As shown in Fig. 7, after applying rules to POS tagged text, the candidate classes, methods, attributes, and relationships are shown to the list. System analysts can select one and/more methods and attributes; also, the class can be passive. In this phase, the class diagram elements have been linked to a requirement, and the traceability matrix generated automatically in an implicit way.

Fig. 3. Proposed methodology.

Fig. 4. Requirements' text [34].

Money Sender Initiate transaction sending SMS containing national ID mobile PIN, mobile number receiver amount money be transferred. Users supposed be mobile number do transactions. User able to send, view, and reply to SMS messages: Money Receiver Receives SMS with authorization key and transaction id, amount, sender mobile Number. Shows the message bank collects money. Bank executives login system, verify customer issue money. The mobile operator receives customer transaction, generates a one-time password, send transaction bank get a reply message bank to send reply time generated password customer. Retailer Check customer validity, check transaction status transaction Receive transaction details via SMS Customer Requests purchase using a cashless purchasing system providing a mobile number, amount, NIC number. Bank Group, Verify Customer, Sends Confirmation message retailer, Sends Confirmation message customer, Sends Retailer. Money Sender User Money Receiver User. Retailer Money Receiver, Customer Money Sender.

Fig. 5. SRS after removing stop words.

Money_NNSender_NNPInitiate_NNPtransaction_NNsendi
ng_VBG SMS_NNP containing_VBGnational_JJ ID_NN
mobile_JJ PIN_NNP number_NN ._,
mobile_JJnumber_NNreceiver_NNamount_NNmoney_NN
be_VBtransferred_VBN ._,
User_NNPsupposed_VBDbe_VBmobile_JJnumber_NNdo
_VBPtraction._NN_ User_NNPable_JJsend_VBP
SMS_NNP message_NNview_NNreply_NN SMS_NNP
message_NN._.Money_NNPReceiver_NNPReceive_NNP
SMS_NNP
message_NNauthorization_NNkey_JJtransaction_NNid_I
Namount._NNsender_VBPmobile_JJ Number._.
Shows_NNSmessage_NNbank_NNcollect_VBPmoney_N
N
Bank_NNPexecutives_NNSlogin_VBPsystem._NNSverify
_VBPCustomer_NNissue_NNmoney_NN ._,
Mobile_NNPoperator_NNreceives_VBZcustomer_NNtran
saction_NNgenerate_VBPone_CDtime_NNpassword_NN
_,
send_VBtransaction_NNbank_NNget_VBreply_VBmessag
e_NNbank_NNsend_VBPreply_NNtime_NNgenerated_V
BDpassword_VBNcustomer_NN ._,
Retailer_NNPcheck_VBcustomer_NNvalidity_NNcheck_
VBPtraction_NNstatus_NNtransaction_NNReceive_VB
Ptraction_NNdetails_NNSvia_IN SMS_NNP
Customer_NNPRequests_NNSpurchase_VBPusing_VBGc
ashless_JJpurchasing_NNsystem_NNproviding_INmobile
_JJnumber_NN ._, amount_NN ._, NIC_NNP number_NN
_._.Bank_NNPgroup_NNVerify_NNPCustomer_NNPSends_
NNPConfirmation_NNPmessage_NNretailer._NNSends_N
NSConfirmation_NNPmessage_NNcustomer._NNSends_
NNSRetailer_NNP ._,
Money_NNPsender_NNPUser_NNPMoney_NNPReceiver
_NNPUser_NNP ._,
Retailer_NNPMoney_NNPReceiver_NNP ._,
Customer_NNMoney_NNSender_NNP ._,

Fig. 6. POS Tagged SRS.

Fig. 7. Selection of class diagram elements (methods, and attributes).

Fig. 8. Selection of class's relationships.

In Fig. 8, the relationships between classes could be selected. We can determine three types of relationships, association, aggregation, and generalization in the proposed tool. The relationship between classes can be automatically determined, but we select using list and form for more flexibility.

Fig. 9. Change Requirements.

In Fig. 9, the requirements need for some change will be selected. We can choose requirements in two ways in our system, either by requirements ID or by the Trace ID. In the traceability matrix, each row is given a unique ID known as trace Id used to manage traceability between a specific class and requirements elements stored in that table. When we select trace id, it will show only specific elements, one element of class or requirements, as Fig. 10 shows.

Fig. 10. Change Requirements Elements through Trace ID.

Fig. 10 displayed after select trace Id as in Fig. 9, the proposed change for requirements elements will be entered and wait for approval (approve or ignore proposed change). It consists of trace Id, Req ID, source, source description, target, and target description. The source is the source of the change either in the design phase or requirements phase. If the change in the requirement phase, the target is the design

phase because it should be updated to become a requirement phase and vice versa. Source description refers to the required element after applying POS Tag, or the name of design elements. Target is the phase that should be reflected by change. Finally, the target description, the element name, and element specify element type (class, attribute, method, or relationship).

Traceability Table						
Enter req_ID		1				
		search				
		Back				
Trace ID	Req ID	Source	Description in So.	Target	Description in Ta.	Design Elements
1	1	req_phase	User	design_phase	User	class
2	1	req_phase	login system.	design_phase	login system.	Method
3	1	req_phase	number receiver	design_phase	number receiver	Attribute
4	1	req_phase	national ID	design_phase	national ID	Attribute
5	1	req_phase	mobile number	design_phase	mobile number	Attribute
6	1	req_phase	Money Sender	design_phase	Money Sender	class
7	1	req_phase	do transaction.	design_phase	do transaction.	Method
8	1	req_phase	send SMS	design_phase	send SMS	Method
9	1	req_phase	view reply	design_phase	view reply	Method
10	1	req_phase	amount money	design_phase	amount money	Attribute
11	1	req_phase	key transaction	design_phase	key transaction	Attribute
12	1	req_phase	Money Receiver	design_phase	Money Receiver	class
13	1	req_phase	Receive SMS m.	design_phase	Receive SMS m.	Method
14	1	req_phase	Shows messag.	design_phase	Shows messag.	Method
15	1	req_phase	amount money	design_phase	amount money	Attribute
16	1	req_phase	mobile PIN	design_phase	mobile PIN	Attribute
17	1	req_phase	Customer	design_phase	Customer	class
18	1	req_phase	Requests purch.	design_phase	Requests purch.	Method
19	1	req_phase	providing mobil.	design_phase	providing mobil.	Method
20	1	req_phase	Mobile operator	design_phase	Mobile operator	class
21	1	req_phase	sending SMS	design_phase	sending SMS	Method
22	1	req_phase	Receive SMS m.	design_phase	Receive SMS m.	Method
23	1	req_phase	login system.	design_phase	login system.	Method

Fig. .11 Traceability Matrix.

Fig. 11 shows the traceability matrix. It consists of a unique Traceability ID (Trace ID) for each requirement/design elements. Requirements ID shows several requirements that the elements belong to. In our case, we assume that the requirements are written as a single text for better processing. Source determines the source of requirements or proposed change; in our case, the default requirements source is the requirement phase, and the target is the design phase. Description in source shows the requirements element in the requirement phase obtained from the processed original text. Target shows the target of the requirement or design phase. Design description is requirements elements that are mapped to the design phase, and design elements specifying elements are either Class, Attribute, Methods, or relationships. This table or matrix handles the necessary information needed in traceability and can obtain many traceability reports. When UML class elements selected all needed traceability, the information will be saved in the traceability table.

V. EVALUATION

To assess the quality and applicability of the developed framework, we perform the following:

1. Make questionnaires that discuss our work to specialists.
2. Mathematical evaluation.
3. RDBT was built; this was discussed in methodology section.

Here we will discuss the evaluation:

1. Questionnaires that discuss our work to specialists.

A survey has been applied and globally distributed to

specialists from LAAS in France, PSAU in Saudi Arabia, SUST in Sudan, NCTR in Sudan under our direct supervision to ensure that we obtain a real evaluation of our work. The survey discusses the framework in details and forms of writing requirements to generate the class diagram elements, and how to manage traceability, and different reports.

Twenty-two persons filled the questionnaire; 59.1% are academics in computer science, 31.8% are software engineers, and 9.1% work in other computer science fields. 54.5% are Ph.D. holders, 31.8% have M.Sc in Computer science or related fields, and 13.6 have B.Sc in computer science or related fields. Most of the respondents are Ph.D. holders in the related fields. Also, we found around 81.8% use Software Requirements Specification(SRS) and Software Design Documents(SDD), while 18.2 did not use SRS and SDD, so the SRS and SDD are used in most cases in the requirements specification and design. It is crucial to have consistency between requirements and design. Regarding this, we mention some problems. One of the significant problems is SRS and SDD consistency. We found that 9.5% strongly agreed that there is a problem. 57.1 agreed that there is a problem. 19% are neutral, and 14.3% disagree that there is a problem. We can say that there is a consistency problem between requirements and design as shown in Fig. 12.

Fig. 12. SRS and SDD consistency.

Requirements can be traced to the design phase to check the consistency. The traceability can be applied manually as Requirements Traceability Matrix(RTM) or automatically. So, it is not easy to maintain full requirements traceability. We found that 4.8% perform it automatically, 38.1% perform it manually, and 57.1% perform semi-automatically performed traceability Fig. 13.

Fig. 13. How consistency requirements consistency applied.

We proposed some rules to write requirements that will facilitate the auto-generation of UML diagrams that allow traceability of requirements and design. As shown in Fig. 14, we found 15% strongly agree that the given rules are easy to understand, while 55% agree the rules are easy to understand, and 30% say it is neutral to understand. This means the given rules, obtained acceptable agree regarding understandability and ease of use.

Fig. 14. Understandability of proposed rules.

The framework was practically implemented using Eclipse IDE, and the payment system was applied as a use case in our developed system. In the survey, the methodology was discussed in detail, and potential questions are put. Does the developed methodology facilitate the process of moving from SRS to SSD consistently? We found that 10% strongly agree, 65% agree, and 25 % are neutral. This reveals that the developed framework is acceptable and facilitates the process of traceability Fig. 15.

Fig. 15. Methodology support SRS to SSD consistency.

Fig. 16. Developed methodology support Bi-directional traceability.

As in Fig. 16, our developed framework developed to support bi-directional traceability, i.e., both backward and forward traceability; 15% strongly agree that the framework support bi-directional traceability, while 75% agree, and 10% are neutral.

1. Mathematical Evaluation:

There is two main mathematical formula used to measure accuracy and completeness[26]:

a. Precision Function:

Reflect the accuracy of produced results and how much produced information, its measure by the following function.

$$\text{Precision} = N_{\text{correct}} / (N_{\text{correct}} + N_{\text{incorrect}})$$

b. Recall Function:

Measures the completeness of produced results and compares the correct and relevant information returned by the solution against what is found by human analysts. It measured by using the following function:

$$\text{Recall} = N_{\text{correct}} / N_{\text{key}}$$

TABLE IV: MATHEMATICAL EVALUATION RESULT

Class Diagram Elements	Class	Attribues	Methods	Relationships
Precision	100%	100%	90%	100%
Recall	100%	100%	100%	100%

As shown in Table IV, the process of the Class diagram was evaluated, and a good result was obtained, high precision and recall in all class elements, because we made our rule engines entirely with patterns/rules that can facilitate the chunking process compared with the studies [27] and [32] which includes a comparison of many previous works. Regarding traceability accuracy, we obtained 100%. All changes are bi-directionally traced and reflected while in [33] they obtained 80%. So, all changes are applied in both requirements and the UML class diagram.

VI. CONCLUSION AND FUTURE WORK

Traceability is one of the crucial issues in software engineering, and especially in requirements. When traditionally moving from SRS to the design phase, this may take time and also inconsistency raised. To solve these issues, we proposed using NLP to generate a UML class diagram semi-automatically from requirements, make bi-directional traceability between two of these stages to make them consistent, and obtain higher quality software. The methodology was applied practically. The use case was applied, evaluated through a well-designed questionnaire approved by two professors. A mathematical evaluation was conducted; all these evaluation results indicate the work is suitable and achieve the research aim. Now machine learning plays a vital role in artificial intelligence. As extensions to this work, we propose to use machine learning and text mining to facilitate the process of patterns writing and generate them through mining techniques.

REFERENCES

- [1] Satyarthi, Sandhya, and Dharendra Pandey. "Framework for Requirement Management using Requirement Traceability", *International Journal of Advanced Research in Computer Science* 8.5 (2017).
- [2] Deeptimahanti, Deva Kumar, and Muhammad Ali Babar. "An automated tool for generating UML models from natural language requirements", 2009 IEEE/ACM International Conference on Automated Software Engineering. IEEE, 2009.
- [3] Systems and software engineering —Life cycle processes — Requirements engineering, INTERNATIONAL STANDARD ISO/IEC/ IEEE 29148, 2018.
- [4] "Traceability Analysis". http://www.chambers.com.au/glossary/traceability_analysis.php, access date July 2018.
- [5] Copestake, A. (2004). Natural Language Processing Lecture Synopsis. *Natural Language Processing*, 39(1), 60–62. Retrieved from <http://www.cl.cam.ac.uk/users/aac/>.
- [6] DHURIA, Shabina. Natural language processing: An approach to parsing and semantic analysis. *International Journal of New Innovations in Engineering and Technology*, 2015.
- [7] ARELLANO, Andres; ZONTEK-CARNEY, Edward; AUSTIN, Mark A. Frameworks for natural language processing of textual requirements. *International Journal On Advances in Systems and Measurements*, 2015, 8: 230-240.
- [8] Amannejad, Yasaman, et al. "From requirements to software design: An automated solution for packaging software classes." *Proceedings of the 2014 IEEE 15th International Conference on Information Reuse and Integration (IEEE IRI 2014)*. IEEE, 2014.
- [9] Gulia, Sarita, and Tanupriya Choudhury. "An efficient automated design to generate UML diagram from Natural Language Specifications." 2016 6th international conference-cloud system and big data engineering (Confluence). IEEE, 2016.
- [10] More, Priyanka, and Rashmi Phalnikar. "Generating UML diagrams from natural language specifications." *International Journal of Applied Information Systems* 1.8 (2012): 19-23.
- [11] Sharma, Richa, Sarita Gulia, and K. K. Biswas. "Automated generation of activity and sequence diagrams from natural language requirements." 2014 9th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE). IEEE, 2014.
- [12] Shinde, Subhash K., Varunakshi Bhojane, and Pranita Mahajan. "Nlp based object oriented analysis and design from requirement specification." *International Journal of Computer Applications* 47.21 (2012).
- [13] Narawita, Chamitha Ramal. "UML Generator-Use Case and Class Diagram Generation from Text Requirements." *ICTer* 10.1 (2017).
- [14] Liu, WenQian, Steve Easterbrook, and John Mylopoulos. "Rule-based detection of inconsistency in UML models." *Workshop on Consistency Problems in UML-Based Software Development*. Vol. 5. 2002.
- [15] Letsholo, Keletso J., Liping Zhao, and Erol-Valeriu Chioasca. "TRAM: A tool for transforming textual requirements into analysis models." 2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE, 2013.
- [16] Essebaa, Imane, and Salima Chantit. "Tool Support to Automate Transformations from SBVR to UML Use Case Diagram." *ENASE*. 2018.
- [17] García, Héctor, Eugenio Santos, and Bruno Windels. "Traceability management architectures supporting total traceability in the context of software engineering." (2008).
- [18] SIMONSSON, MIKAEL. "Introducing a New Framework for Modelling Requirements, Traceability, and Patterns in UML." (2013).
- [19] Chanda, Jayeeta, et al. "Traceability of requirements and consistency verification of UML use case, activity and Class diagram: A Formal approach." 2009 Proceeding of International Conference on Methods and Models in Computer Science (ICM2CS). IEEE, 2009.
- [20] Mader, Patrick, Orlena Gotel, and Ilka Philippow. "Getting back to basics: Promoting the use of a traceability information model in practice." 2009 ICSE Workshop on Traceability in Emerging Forms of Software Engineering. IEEE, 2009.
- [21] Min, Hyun-Seok. "Traceability guideline for software requirements and UML design." *International Journal of Software Engineering and Knowledge Engineering* 26.01 (2016): 87-113.
- [22] Noyer, Arne, et al. "Traceability and interfacing between requirements engineering and UML domains using the standardized ReqIF format." 2015 3rd International Conference on Model-Driven Engineering and Software Development (MODELSWARD). IEEE, 2015.
- [23] Kaufmann, Andreas, and Dirk Riehle. "Improving traceability of requirements through qualitative data analysis." *Software-engineering and management* 2015 (2015).
- [24] Plataniotis, Georgios, et al. "Traceability and modeling of requirements in enterprise architecture from a design rationale perspective." 2015 IEEE 9th International Conference on Research Challenges in Information Science (RCIS). IEEE, 2015.
- [25] NLP Guide: Identifying Part of Speech Tags using Conditional Random Fields, Retrieved from <https://medium.com/analytics-vidhya/pos-tagging-using-conditional-random-fields-92077e5eaa31>
- [26] Sharma, Richa, Pratyoush K. Srivastava, and Kanad K. Biswas. "From natural language requirements to UML class diagrams." 2015 IEEE Second International Workshop on Artificial Intelligence for Requirements Engineering (AIRE). IEEE, 2015.
- [27] Limón, Angelina Espinoza. An advanced traceability schema as a baseline to improve supporting lifecycle processes. Diss. Universidad Politécnica de Madrid, 2009.
- [28] Li, Xiaoshan, Zhiming Liu, and Jifeng He. "Consistency checking of UML requirements." 10th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'05). IEEE, 2005.
- [29] Khan, Ali Hanzala, and Ivan Porres. "Consistency of UML class, object and statechart diagrams using ontology reasoners." *Journal of Visual Languages & Computing* 26 (2015): 42-65.
- [30] Yang, Yilong, et al. "Integrating UML with service refinement for requirements modeling and analysis." *IEEE Access* 7 (2019): 11599-11612.
- [31] "Analysis and designing of the automated mobile money transfer system", chapter 5. Available: <http://dl.lib.mrt.ac.lk/bitstream/handle/123/1785/Chapter05.pdf?sequence=5>.
- [32] Harmain, Harmain Mohamed, and R. Gaizauskas. "CM-Builder: an automated NL-based CASE tool." *Proceedings ASE 2000*. Fifteenth IEEE International Conference on Automated Software Engineering. IEEE, 2000.
- [33] Haouam, Mohamed Yassine, and Djamel Meslati. "Towards automated traceability maintenance in model driven engineering." *IAENG International Journal of Computer Science* 43.2 (2016): 147-155.
- [34] Dawood, Omer Salih. "Toward Requirements and Design Traceability Using Natural Language Processing." *European Journal of Engineering Research and Science* 3.7 (2018): 42-49.