From Requirements Engineering to UML using Natural Language Processing – Survey Study
Omer Salih Dawood, Abd-El-Kader Sahraoui

To cite this version:


HAL Id: hal-01703317
https://hal.laas.fr/hal-01703317
Submitted on 7 Feb 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
From Requirements Engineering to UML using Natural Language Processing – Survey Study

Omer Salih Dawood¹ and Abd-El-Kader Sahraoui²
¹ Sudan University of Science and Technology, Khartum, Sudan
² LAAS-CNRS, Université de Toulouse, CNRS, UT2J, Toulouse, France
¹ omercomail@gmail.com

Abstract— In the paper process of moving from software requirements to Unified Modeling Language (UML) diagrams has been studied. It shows the importance of this process and discusses many comparative studies in the field. A questionnaire related to the study was distributed worldwide to many research groups, academia, and industry to know the current status of using requirement management tools, knowledge of using UML in software development, frequently used UML diagrams, and the methodology used to generate UML diagrams from requirements. The paper emphasises that there is a need to do some important research in the area of generating diagrams from requirements. Section 2 concludes and compares the research work in the area of TEXT to UML, and introduces requirements management tool. Section 3 explain the Survey Study, and section 4 Discuss the survey results.

A. Unified Modelling Language (UML)

UML is de facto standard for modeling of software. Object Management Group (OMG) was standardized UML in 1997 [16]. It is unification of three methods Booch, OMT, and OOSE.

According to [17] the UML has much type of diagrams, use case diagrams, and behavioral diagrams which include state diagram, activity diagram, sequence diagram, and collaboration diagram, and the other one is static diagrams like class diagram. It’s a language for specifying, visualizing, constructing and documenting software systems. It’s not just modeling software, many domains can be modeled UML like System Engineering, Process Modeling, and representing the organizational structures [18].

B. Requirements Engineering:

According to SEI Requirements engineering confirms systematical and repeatable using of techniques ensure the completeness, consistency, and relevance of the system requirements [19]. It’s very complex process because it involves the requester, developer, and author. The requesters know what they want but they don’t know how to develop a system, while developer knows how to develop a system but they don’t know what is the problem, the author tries to minimize communication gabs between requester and developer[19].There are two types of requirements, as follows:

1. User requirements: describe the services expected from system and constraints that the system should follow. It must be written in statements that the non-technical person can understand [20].

2. System requirements: describe more and deep details of user requirements. Software engineers analyze these requirements to know what exactly implement in the system. System requirements include both functional and non-functional requirements [20].
NLP is processing of human natural language automatically or semi-automatic. NLP is essentially multidisciplinary and related with linguistics[22]. NLP and SE are both, the branches of computer science. In Software Development Life Cycle can be applied to every phase. There are textual artefacts in analysis and design phases like Requirement Document and Software Design Specification. Quality attributes like performance, feature, reliability, aesthetics, and perception are need to be assessed in NLP software [14]. Most of the requirements are written in natural language text and this causes many issues like ambiguity, specification issues, and incompleteness. These requirement statements need to be analysed then we need to use natural language processing for that purpose because NLP provides many tools which help in linguistic analysis and helps in automated assistance [15]. Using NLP in RE is important because the requirement specification is written in cooperation between software house analyst and users and customers, and the customers will not sign contract if requirements are written in formal language[15].

II. UML GENERATION FROM REQUIREMENTS USING NLP

Priyanka and Rashmi [1] developed a tool for UML diagrams generation from requirements using NLP. The tool is known as (RAPID), which is used to analyse textual requirements, finding core concepts and its relationships, and finally generate UML diagram [1].

RAPID Consists of OpenNLP parser, RACE stemming algorithm, and Word Net, Domain Ontology module is used to improve the performance of concepts identification and Class Extraction Engine module receives the output of “concept extraction engine” module and applies different heuristic rules to generate both classes, relationship, and attributes[1].

Deva and Muhammad [2] developed a tool named as UML Model Generator from Analysis of Requirements (UMGAR) used to generate UML models from requirements using NLP tools, generated UML models like the Use-case Diagram, Analysis class model, Collaboration diagram and Design class model. They combined both Rational Unified Process (RUP) to specify objects, and ICONIX process that uses most a good NLP tools. There are two main components that composed UMGAR process architecture as follows:

1. Normalize requirements component (NLP Tool Layer), used to identify requirements ambiguity and incompleteness and It consists of many sub components including Syntactic Reconstruction sub component uses 8 syntactic reconstruction rules.

2. Model Generator component used to generates many OO models from normalized requirements consists of Use-case Model Developer, and class model analysis and class model developer.

Andres and et al [3] developed a new approach and tool that interpret, organize, and manage requirements through application-specific ontologies and natural language processing. They used tool known as Natural Language Toolkit (NLTK). The tool receives raw requirement text and performs segmentation to the entered text to obtain sentences, after that text will enter to tokenization process to tokenize text into words or punctuation character and normalize these tokens through stemming process, then perform par of speech tagging to identify the role of each word, entities can identify On top of POS tagging, then identify group tokens specially parsed words that represent the entity through chunking process, and finally recognize requirements[3]. NURI and et al [4] develop an approach to generate behavioral diagrams from user requirements they focused on both use case diagram and activity diagram. The developed tool named as RETRANS (REquirement TRANSformation). The approach receives the requirement and performs POS using Stanford POS Tagger, after that there are phases as follows:

Phase 1: requirement phase, First Phase is requirements keywords tracking used to detect the actor and use cases for the requirement. Phase 2: Use Case Diagram generation, the use case diagram library is used to generate use case diagram by connecting actors with related use case. Phase 3: Activity Diagram generation, after use case generation. The system automatically specifies all use cases that involved in the use case diagram, then it will generate the activity diagram by linking the components inside the class library (activity).

Prasanth Nakul[5] addresses the interdisciplinary between software engineering and natural language processing, and proposed methodology known as TextToUml to produce high-quality UML diagrams. The methodology consists of five diagrams as follows:

1- Define N.L. text quality parameters, and this includes classifying text to controlled language and uncontrolled language.

2- Identify level of noise and complexity of the text and classify sentences into simple, semi-complex, and complex.

3- Identify the type of diagram according to the description given in the text.

4- Determine UML diagram specification.

5- Derive UML specification with N.L. text tuning to all available UML diagrams.

6- To generate UML diagrams, build interface between the ontology and application [5]

In [6] S.G. MacDonell and et al developed architecture of an autonomous requirements specification by using a natural language processing (NLP). They focused on the verification of requirements specification analysis.
As shown in figure (2) the system architecture consists of three modules as follows:
1- Tokeniser – reads requirements from a document
2- Parser - parses requirements sentence to extracts all unique noun terms.
3- Term management system – used to filter unimportant terms, classify remaining terms (function, entity, or attribute), and insert the object into a project knowledge base.

Subhash and et al [7] built a tool that analyzes requirement texts and builds model of the processed text represented in the semantic network. Their tool consists of two modules, NL analysis and diagram & code generation [7]. In the first module POS tagging is used to analyze and classify tokens, then the text understanding categorizes text into more further classes, as object, messages, verb, and etc to facilitate class generation process, Knowledge extraction module receives the output of previous phases to extract classes, attributes, and actors. Finally the UML diagrams will be generated. Based on generated UML diagrams, and extracted knowledge, Second module generates code in language like java.

In Deva and Ratna [8] proposed a tool named as Static UML Model Generator from Analysis of Requirements (SUGAR) to produce static UML models from natural language. The SUGAR receives requirement texts and split complex sentences into simple sentences using syntactic reconstruction rules, then parse requirement document and generate parse tree by using Stanford Parser which the subjects and predicates are identified, actors mostly are subjects, and use cases are mostly the predicates, then identify the relationships between use cases and actors, then draw use case diagram. Class diagram can be generated by using the previous items and identify classes, methods, attributes, and relationships. Yasaman and et al [9] automated the process of generating package diagram in software design to facilitate design process and because process of moving from requirements engineering to design done by ad-hoc way. There are three components, first is static view generator - static ontology used to generate knowledge about the system. In the static ontology, classes with similar functionalities are grouped together. Second is dynamic view generator - dynamic view of ontology shows the interactions between the systems. Third is package diagram recommender that is used to receive the output of static and dynamic ontology generator to generate package diagram. A hierarchical clustering algorithm was used as core of packing solution. The classes are grouped together in same packages if they have higher number of communications and similar communication pattern between them [9].

Vibhu and et al [10] proposed a technique to generate high level class diagram from requirements, they implemented the approach into Functional Design Creation Tool (FDCT). They used heuristic rules and domain specific glossary to create the design. They developed Requirements Analysis Tool (RAT) that were used for restricts the requirement sentence, and perform lexical and semantic analysis on requirement document. RAT classifies requirements into six types which cover set of wide rage application requirement types. There are three phases approach for RAT to analyze requirements, in the first phase, requirements statement convert into a set of tokens with the user help to define glossaries. In the second phase the state machines are used for analysis of the requirement statements' syntax [10]. The third phase consists of semantic analysis with the help of domain specific ontology [10]. Sarita and Tanupriya [11] proposed an algorithm to automatically generate UML diagrams from user requirements after receiving requirements text, the tool is tokenized and Pos tagger is used to perform lexical tagging, then extract verbs and objects as an activity, finally generate activity diagram. To generate sequence diagram, after receiving requirements the plain text file is pre-processed, then the parser defines the structure of sentence [11].

Imran and et al [12] developed architecture to generate UML class diagram as in figure (3).
more (Conjunction, Preposition, Precondition and Marker). There are four types of frame structure, active, passive, Conjunction between verbs with Passive Voice, and Preposition. The requirement statement is tokenized and belongs to specific frame, then the object, action, relationships can be identified [13].

The questionnaire aims to answer the following questions:
1. What is familiarity of Formal methods, SADT (Structured Analysis and Design Technique), and OMT (Object Modelling Technique)?
2. Do the organizations/system engineers use a systematic way to gather and document requirements and familiarity in using requirements management tool?
3. How organizations/system engineers generate UML diagrams from requirements?
4. Are organizations/system engineers need tools/techniques that facilitate the process of moving from requirements engineering stage to software design stage?
5. The order of using UML diagrams - what is most used and needed UML diagrams.

### TABLE I: Comparison Table

<table>
<thead>
<tr>
<th>Study</th>
<th>Generated Diagrams</th>
<th>Main Components</th>
<th>Strengths</th>
<th>Weaknesses</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1]</td>
<td>Class diagram</td>
<td>RAPID Concept Extraction Engine + RAPID Class Extraction Engine + some small components</td>
<td>Deeply shows the details of class elements generation</td>
<td>Didn't generate class code + only concentrates in one diagram generation</td>
</tr>
<tr>
<td>[2]</td>
<td>use-case + analysis class models + collaboration diagram</td>
<td>Normalizing requirements component + Model Generator component</td>
<td>Generates three UML diagrams</td>
<td>Need some additives to generate more diagrams</td>
</tr>
<tr>
<td>[3]</td>
<td>Class Diagram</td>
<td>Segmentation + Tokenization + POS + entity and relation recognition</td>
<td>Uses ontology to assist in diagram generation and good in identifying relationships</td>
<td>Only class diagram generation without code. Needs some additives to cover many diagrams.</td>
</tr>
<tr>
<td>[4]</td>
<td>Use case and Activity diagram</td>
<td>NLP + use case library + activity diagram library</td>
<td>Generate dynamic diagrams</td>
<td>Some enhances are needed to generate more diagrams</td>
</tr>
<tr>
<td>[5]</td>
<td>Not specified</td>
<td>Many steps in algorithm</td>
<td>General diagram generation</td>
<td>Didn't show practical work</td>
</tr>
<tr>
<td>[6]</td>
<td>Class diagram</td>
<td>NLP tool + Term Management System</td>
<td>Extract complex sentence</td>
<td>One diagram is few, need to generate more</td>
</tr>
<tr>
<td>[7]</td>
<td>class diagram</td>
<td>Natural Language Analysis Block + Diagrams &amp; Code Generation Block Generator</td>
<td>Generate both static and dynamic diagrams - use case and class diagram and java class code</td>
<td>Some enhances are needed to generate more static and dynamic diagrams</td>
</tr>
<tr>
<td>[8]</td>
<td>Use case and class diagram</td>
<td>Use-case Model Generator + Class Model Generator</td>
<td>Generate both static and dynamic diagrams - use case and class diagram and java class code</td>
<td>Some enhances are needed to generate more diagrams</td>
</tr>
<tr>
<td>[9]</td>
<td>Generate package diagram</td>
<td>Static Ontology Generator + Dynamic Ontology Generator</td>
<td>Using ontology and Good in grouping classes together</td>
<td>One diagram is few, need to generate more</td>
</tr>
<tr>
<td>[10]</td>
<td>class diagram</td>
<td>RAT + Tokens + heuristic rules + UML creator module</td>
<td>Generate a high-level class diagram in a good methodology</td>
<td>One diagram is few, need to generate more; didn’t generate classes code</td>
</tr>
<tr>
<td>[11]</td>
<td>Activity diagram + sequence diagram</td>
<td>Activity: Sentence splitter + POS tagger + verb and object extractor. Sequence: pre-processing + parser + additional information identifier + adding conditions</td>
<td>Generation both activity and sequence diagrams in good manner</td>
<td>Concentrates only in some dynamic diagram</td>
</tr>
<tr>
<td>[12]</td>
<td>Class diagram</td>
<td>Natural Language Analysis Block (POS tagging + Text Understanding) + Knowledge Extraction + Diagram and Code Generation Block [ UML diagrams] + Code Generation</td>
<td>Generate UML class diagram in a good way with class’s code.</td>
<td>Only one static class diagrams are generated, need more enhancement to generate more classes</td>
</tr>
<tr>
<td>[13]</td>
<td>Activity diagram + sequence diagram</td>
<td>Stanford POS tagger + Frame structure (Active Voice, Passive Voice, Conjunction between Verbs with Passive Voice, and Preposition)</td>
<td>Classify statements into simple and complex statements. Frame is a good idea to generate both activity and sequence diagram</td>
<td>They assume that no redundancy and ambiguity. Need some enhancement to cover more diagrams</td>
</tr>
</tbody>
</table>

### III. METHODOLOGY:

The research aimed to improve the process of generating UML diagrams from requirements, many related work are reviewed firstly to specify the weaknesses and what is the problem in this area, to know the range of using requirements in organized way and how the organizations generate UML from requirements, to know all these a Questionnaire was developed by consulting many computer science professors and distributed worldwide to many IT companies, universities, and research groups to ensure the questionnaire covers wide range, and allow to obtain good and common result. Both academia and industry that related with software and system engineering filled the questionnaire, we expected more than 100 persons will fill the questionnaire but only around 92 persons filled the questionnaire, and good results obtained and the research achieve the objectives. The questionnaire was distributed and filled during the period from 01 November to 01 December 2016. The questionnaire has many sections each...
IV. RESULTS AND DISCUSSIONS

The questionnaire was distributed worldwide and filled by 92 respondents from academia, industry, or both we found that around 25% weren’t familiar with Formal methods, SADT (Structured Analysis and Design Technique), and OMT (Object Modelling Technique), and around 75% familiar with one or more from these concepts. There are 92% familiar with basic UML concepts. The survey found that 69% is using a systematic methodology for collecting requirements, while 31% collect requirements without systematic methodology, and around 67% of the organizations follow Software Development Life Cycle to develop systems. Regarding knowledge of using requirement management tools we found that 17.8% has strong knowledge, while 45.6% has medium knowledge, and 36.7 has poor knowledge. This means that there is need to encourage the organizations/system engineers to use requirement management tools. It is very important to generate UML diagrams from requirements by aiding of tool to ensure all requirements are covered into design then obtain high quality software, as shown in figure (4), 40.2% organizations/system engineers generate UML diagrams manually, while 23.9% generate UML diagrams by semi-automatic methodology, 13% generate UML diagrams by automatic method, and 22.8% didn't generate UML diagrams from requirements. Manual process of generation of UML from requirements needs some enhancement to become automatic or semi-automatic.

UML has many types of diagrams each one has specific role. Figure (4) shows the order of using UML diagrams, the use case diagram, class diagram, sequence diagram, and activity diagram has high percentage use respectively. It's necessary to help generation of these diagrams according to the order of usage.

Figure (4): Methodology of UML generation

V. CONCLUSION:

The paper reviewed the current status of using Natural language processing in software engineering to process the software requirements to generate UML diagrams. The paper deeply studied many research in this area and made a comparison between them and identified each weaknesses and strength and scope for improvement.

A Questionnaire was distributed world wide about using UML, requirement management tools, and how the software engineers generate UML diagrams from requirements. The main purpose of this questionnaire was to enhance software quality and Minimize the design time, cost, and error with reducing human intervention in the design phase through improving the process of generating UML diagrams from requirements using NLP.

We found that only around 13% of users/organizations generate UML diagrams automatically and around 23.9 using semi-automatic way to generate UML diagrams, while 40.2% generate UML diagrams manually, and this means that there is need to minimize this percentage by more studies and enhancing in UML generation from requirements automatically or semi automatic way.

REFERENCES


Ann Copestake, “Natural Language Processing,” Lecture Synopsis , 2004