International Journal of Engineering & Technology, 7 (4.19) (2018) 168-171

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research paper

The Role of Natural Language Processing in Requirement Engineering

Hussin Ahmed[1], Azham Hussain[2*], Fauziah Baharom[3]

Human-Centered Computing Research Lab, School of Computing, Universiti Utara Malaysia, 06000, Malaysia
**Corresponding Author E-mail: [2]azham.h@uum.edu.my*

Abstract

The major objective of Software Requirements Specification (SRS) is providing sufficient information for software developers to build software product successfully. However, the current features of natural language hinders processing and analysis of requirements due to its ambiguous nature. Over the years, many Natural Language Processing (NLP) approaches were emerged to tackle this problem to detect errors or extract useful information from requirements documents. In this paper, a review of these approaches has been represented to reveal the role of NLP in requirement engineering and depict the current dilemma of SRS processing.

Keywords: *natural language processing, software requirements specification, requirements methods*

1. Introduction

A successful software product depends largely on how well the requirements have been understood and transformed into relevant functionalities in the software(Shah & Jinwala, 2015). A requirement can be defined as a property that a system must exhibit for meeting the system's motivating need (Dube & Dixit, 2010).Requirements specification is the process of documenting user and system requirements (Robie, Baharom, & Mohd, 2014). The output of requirements specification is SRS document which serves as a fundamental repository of all requirements stated by customers.Furthermore, it is indispensable reference to every stage in Software Development Life Cycle (SDLC). Mapping functional requirements first to specifications and then to code is a challenging task in software engineering(Diamantopoulos et al., 2017). Hence, SRS has to provide sufficient information for software developers in order to gain a deep understanding about the proposed functionalities of software. A SRS document tends to follow a previously defined template (see Figure 1). This template represents the structure of the document including chapters and sections and equipped with supplementary practical guidelines (Rodrigues et al., 2014). Nigam et al. (2012) stated that SRS is the primary vehicle for agreement between the software developers and customers and it is the basis for judging fulfilment of contractual obligations. As a consequence, this agreement has to reflect a clear understanding of the requirements in unambiguous manner.

Fig. 1: Software requirements specification outline (ISO/IEC/IEEE, 2011)

Requirements are considered as an input to design, implementation and validation phase of software product development (Hussain, Mkpojiogu, & Kamal, 2016). For this reason, the ongoing concern in writing a requirements document is to make a balance between two important aspects, the need to make the requirements amenable to processing and the

need to make the requirements document in a readable format (Hull et al., 2010).
Achieving this balance will help to avoid miscommunication among stakeholders and misinterpretation of requirements. Moreover, this balance will facilitate and accelerate the agreement between the customers and software developers regarding implementing what is written in SRS. The acceleration of agreement will lead to higher productivity in SDLC and contribute to reduce time and decrease cost. As a result, the specification language has to fulfill the aforementioned two criteria for increasing the productivity and producing precise query to software developers.

2. Literature Review

Since, natural language is widely understood by stakeholders, it is used as a common way for representing requirements. Representing requirements in natural languagesuffers from potential problems like ambiguity, inconsistency and incompleteness. A systematic literature review in the last two decades from 1995 till 2016 shows that collecting ambiguous requirements is one of the highest critical challenges in software engineering (Bin et al., 2016).Since the advent of software engineering, researchers used formal and semi-formal methods to overcome this problem. However, even when formal and semi-formal languages are used, there is no escape from natural language as the initial requirements are written in natural language (Kamsties, 2005). The consequences of ambiguous requirements will lead to excessive efforts, high cost and failure in some software projects. For example, software developers might decide a subjective interpretation of requirements based on their point of view. Ferrari et al., (2014) argued that this subjective interpretation leads to designing software in a different way from what was intended in the requirements.
For several decades, SRS processing and analysis has been the focus of research in software engineering discipline. Since natural language is ambiguous, a computer cannot provide full support to analyze SRS in an automatic fashion. Consequently, the analysis of SRS is conducted manually which consumes time, effort and cost. Most importantly, the manual analysis of requirementsresults in inefficiency and imprecise results (Wang, 2016). The problem will be more obvious and critical when software projects involve thousands of requirements and hundreds of SRS documents. Conducting verification of thousands of requirements via humans will become extremely expensive (Fanmuy et al., 2014). Generally, the primary source of problems in requirement engineering is reliance on humans extensively (Ahmed, 2018). This discussion leads to the importance of finding an automatic way for processing SRS.NLP was used as a possible solution to resolve ambiguity and to provide valuable information to the intended software developers. Ryan (1993) argued that:" *It is highly questionable that the resulting system from NLP would be of great use in requirements engineering*". Nazir et al. (2017) conducted a systematic literature review on NLP applications for software requirement engineering, he concluded that: "*Manual operations are still required on initial plain text of software requirements before applying the desired NLP techniques*". In this paper, a complementary review is conducted to understand the role of NLP in the context of requirement engineering.

3. Methodology

A review of multiple resources in current literature is conducted to answer the questions of this paper. The objective of this study is to answer the following two questions:
Q1: What are the current practices ofNLP in SRS processing?
Q2: What are the current limitations of NLP in SRS processing?

4. Current Practices of Software Requirements Specification Processing

Abbott (1983) has been credited with his pioneering work in developing an informal strategy to derive the output as per object oriented concepts. In this strategy, data types are suggested by common nouns, objects are referenced by proper *noun* and *reference*, control structures are suggested by using *if, then, else, for, do, until and when*. (Booch, 1986) developed a method that extended Abbott's approach and emphasized on creating an interface of object to draw a boundary between the inside view and the outside view of object. Abbott (1983) and Booch (1986) concluded that it is not a purely mechanical process to transform their informal strategy into a formal program and it requires a great deal of background knowledge for the process of transformations. This means that human intervention was still necessary to identify words that are suitable for generating object models.
Currently, researchers strived to extract information from SRS by developing methods that rely on NLP. NLP is the computerized approach to analyzing text and being a very active area of research and development (Reshamwala et al., 2013). Current approaches use many techniques to achieve two main objectives. The first objective is to detect defects in SRS such as Körner and Brumm (2009), Huertas (2012) and Rago et al. (2016). The second objective is to generate Unified Modeling Language (UML) diagrams from SRS such as Ilieva and Ormandjieva (2005), Kothari (2012), MacDonell et al. (2014), Landhaußer et al. (2014), Elallaoui et al. (2015), Gulia and Choudhury (2016), Iqbal and Bajwa (2016), Vemuri et al. (2017) and Diamantopoulos et al. (2017).
The vast majority of current approaches relied upon unrestricted natural language in combination with using Part of Speech (POS) technique. The essential purpose of POS is to conduct a semantic analysis which includes assignment of one or more tags to each word in a sentence (Tripathy & Rath, 2014). These tags identify the grammatical category of each word and consider the categories of words as counterparts to other terminologies in object oriented language. For instance, *nouns* represent objects and *verbs* represent functions. Ilieva and Ormandjieva (2005)proposed a methodology via using POS to organize sentences in groups and built a semantic network from these groups to transform requirements into object oriented model. Huertas (2012) developed a tool called Natural Language Automatic Requirement Evaluator (NLARE). This tool used a set of defined rules as well as regular expressions to look for problems like ambiguity, incompleteness, and atomicity on functional requirements specifications. Kothari (2012) developed a tool called Natural language Processing for Class (NLPC) to obtain basic elements of a class diagram from natural language requirements. MacDonell et al. (2014) developed a system which is composed of three modules with a user interface. The functionality of this system relies upon syntax parsing each sentence by NLP tool to extracts all unique *noun* terms.A drawback of this system is that the syntactic parser can produce ambiguous parse trees of each sentence.
Some approaches relied on Semantic Role Labeling (SRL) in addition to POS to assign relations in sentences (i.e., who did what to whom) such as Körner and Brumm (2009) and Landhaußer, Korner and Tichy (2014). Körner and Brumm (2009)created a tool called Requirements Engineering Specification Improver (RESI). The main objective of RESI is to support requirement analysts via checking for linguistic defects in specifications and to offer a dialog-box to make suggestions for improving the text. RESI made use of NLP tools in addition to ontological reasoning for detecting linguistic defects like distortion and incompletely specified process words. Landhaußer, Korner and Tichy (2014) extended the functionality of RESI via automatic UML models generation and change impact analysis. An obvious limitation of

using SRL is the generality of semantic representation (Ludwig et al., 2018). This generality cannot produce efficient information in the context of sophisticated discipline like software engineering. Elallaoui et al. (2015) created syntax of user stories and used POS for extracting the primary elements of every user story like *Actor, Action and Benefit*. Then, an algorithm was developed to generate the sequence diagrams as an output for every sentence in user stories. Rago et al. (2016) developed an approach called Requirement analyzer with sequence Aligner (ReqAligner). The purpose of this approach is to aid analysts to detect duplicate functionalities in use cases in an automated fashion. Gulia and Choudhury (2016) focused on generating sequence diagram and activity diagram from requirements specification. In this approach, POS technique was used to tag words and two algorithms were developed to generate activity diagram and sequence diagram. Vemuri et al. (2017) developed a tool for learning from patterns in requirements and applying a probabilistic approach in order to simplify identification of actors. Diamantopoulos et al. (2017) used many techniques in conjunction with POS like tokenization, lemmatization and dependency parsing to semantically annotating functional requirements. Tokenization separates the sentence to identify tokens, lemmitization determines the uninflected base forms of words and dependency parsing determines the grammatical relations that exist between two words.

Although NLP has many advantages in many fields, it is not efficient in the context of SRS. Taking into consideration that NLP algorithms are restricted by processing only the information that they can see (Cambria & White, 2014). In a similar vein, Gupta et al. (2013) declared that POS technique cannot produce a precise query in the context of software engineering. This is quite true, since the functionality of POS is to discover the grammatical category of each word in specification written in natural language, it is difficult to represent the exact terminologies in object oriented programming which differs from the general usage of unrestricted natural language. Normally, customers have a great deal of freedom to express their needs without restriction on their use of natural language. As a result, there is undesirable consequence resulted from using unrestricted natural language. In essence, the final analysis of natural language sentence using NLP may not give correct result (Osborne & Macnish, 1996). Also, this analysis produces inaccurate or incomplete models that need validation and extensive manual revision (Selway et al., 2015).

On the other hand, Iqbal and Bajwa (2016) developed a method to generate UML activity diagram from Semantics of Business Vocabulary and Rules (SBVR). SBVR is created by Object Management Group (OMG) as a standard to produce controlled representation of English language for documenting business specifications (OMG, 2017). However using SBVR is still an onerous task to define a complete set of rules and concepts governing a business (Selway et al., 2015;Nemuraite et al., 2010). Selway et al. (2015) noticed that manual interpretation of the business specification written in SBVR is required and the involvement of technical experts remains necessary. Wang (2016) and Landhaußer et al. (2014) reported that using controlled language still not practical for using it in SRS. This was driven by the difficulty of applying controlled language in existing SRS documents and the limited freedom of representation of requirements.

5. CurrentLimitations of Natural Language Processing

Although, there has been a substantial amount of research concerning using NLP in software engineering, there are still critical limitations. Bano (2015) conducted a mapping study and stated the following observations. Firstly, the software engineering research community has not paid enough attention regarding the empirical evaluation of NLP tools and techniques for addressing ambiguity in requirements. Secondly, the researchers have focused more on detection of ambiguity whereas avoiding or resolving ambiguity has been largely neglected in empirical work (Bano, 2015). Moreover, Umber and Bajwa (2011) argued that there is no appropriate approach or tool for providing an automatic mean of removing or minimizing ambiguity in natural language. From the aforementioned observations, it is quite evident that ambiguity has to be handled initially via representation of requirements in a machine-readable format. This representation will ensure consistent use of terminologies and results in accurate transformation of requirements into models.

6. Conclusion

NLP does not have the capability to produce efficient and reliable result that can encourage software community to make more investment in this research area. In order to use NLP efficiently, it is important to use a controlled natural language with comprehensive syntax and predefined static semantics. Developing controlled language in a usable way with the assistance of interactive tools will be highly useful to requirement engineer and help to reduce time of repetitive work. Most importantly, this will helpto extract information precisely andproduce high-quality software product.

References

[1] Abbott, R. J. (1983). Program Design by Informal English Descriptions. Communications of the ACM, 26(11), 882–894.

[2] Ahmed, U. (2018). A review on knowledge management in requirements engineering. In In Engineering and Emerging Technologies (ICEET), 2018 International Conference (pp. 1–5).

[3] Bano, M. (2015). Addressing the challenges of requirements ambiguity: A review of empirical literature. In In Empirical Requirements Engineering (EmpiRE), 2015 IEEE Fifth International Workshop (pp. 21–24). IEEE.

[4] Bin, L., Rahim, A. B., Dominic, P. D. D., Besrour, S., Bin, L., Rahim, A. B., & Dominic, P. D. D. (2016). A quantitative study to identify critical requirement engineering challenges in the context of small and medium software enterprise. In In Computer and Information Sciences (ICCOINS), (pp. 606–610).

[5] Booch, G. (1986). Object-oriented development. IEEE Transactions on Software Engineering, 211–221.

[6] Cambria, E., & White, B. (2014). Jumping NLP curves: A review of natural language processing research. IEEE Computational Intelligence Magazine, 48–57.

[7] Diamantopoulos, T., Roth, M., Symeonidis, A., & Klein, E. (2017). Software requirements as an application domain for natural language processing. Language Resources and Evaluation, 51(2), 495–524. http://doi.org/10.1007/s10579-017-9381-z

[8] Dube, R., & Dixit, S. K. (2010). Process-oriented complete requirement engineering cycle for generic projects. In In Proceedings of the International Conference and Workshop on Emerging Trends in Technology (pp. 194–197).

[9] Elallaoui, M., NAFIL, K., & TOUAHNI, R. (2015). Automatic generation of UML sequence diagrams from user stories in Scrum process. In In Intelligent Systems: Theories and Applications (SITA), 2015 10th International Conference (pp. 1–6). IEEE.

[10] Fanmuy, G., Fraga, A., & Llorens, J. (2014). Requirements verification in the industry. In In Complex Systems Design & Management (pp. 145–160). Springer, Berlin, Heidelberg. http://doi.org/10.1007/978-3-642-25203-7

[11] Ferrari, A., Lipari, G., Gnesi, S., & Spagnolo, G. O. (2014). Pragmatic ambiguity detection in natural language requirements. In In Artificial intelligence for requirements engineering (AIRE), 2014 IEEE 1st International Workshop (pp. 1–8). IEEE.

[12] Gulia, S., & Choudhury, T. (2016). An efficient automated design to generate UML diagram from Natural Language Specifications. In In Cloud System and Big Data Engineering (Confluence), 2016 6th International Conference (pp. 641–648).

[13] Gupta, S., Malik, S., Pollock, L., & Vijay-Shanker, K. (2013). Part-of-speech tagging of program identifiers for improved text-based software engineering tools. In In Program Comprehension (ICPC), 2013 IEEE 21st International Conference (pp. 3–12).

[14] Huertas, C. (2012). NLARE, a natural language processing tool for automatic requirements evaluation. In In Proceedings of the CUBE International Information Technology Conference (pp. 371–378). ACM.
[15] Hull, E., Jackson, K., & Jeremy, D. (2010). Requirements Engineering. Springer Science & Business Media.
[16] Hussain, A., Mkpojiogu, E. O., & Kamal, F. M. (2016). The role of requirements in the success or failure of software projects. International Review of Management and Marketing, 6, 306–311.
[17] Ilieva, M. G., & Ormandjieva, O. (2005). Automatic transition of natural language software requirements specification into formal presentation. In In International Conference on Application of Natural Language to Information Systems (pp. 392–397). Springer.
[18] Iqbal, U., & Bajwa, I. S. (2016). Generating UML activity diagram from SBVR rules. In In Innovative Computing Technology (INTECH), 2016 Sixth International Conference (pp. 216–219).
[19] ISO/IEC/IEEE. (2011). 29148-2011 - ISO/IEC/IEEE International Standard - Systems and software engineering -- Life cycle processes --Requirements engineering. Retrieved from https://ieeexplore.ieee.org/document/6146379/
[20] Kamsties, E. (2005). 11 Understanding Ambiguity in Requirements Engineering. In In Engineering and Managing Software Requirements (pp. 245–266). Springer, Berlin, Heidelberg.
[21] Körner, S. J., & Brumm, T. (2009). Natural language specification improvement with ontologies. International Journal of Semantic Computing, 3(4), 445–470. http://doi.org/10.1142/S1793351X09000872
[22] Kothari, P. R. (2012). Processing natural language requirement to extract basic elements of a class. International Journal of Applied Information Systems (IJAIS), 3(7), 39–42.
[23] Landhaußer, M., Korner, S. J., & Tichy, W. F. (2014). From requirements to UML models and back: how automatic processing of text can support requirements engineering. Software Quality Journal, 22(1), 121–149. http://doi.org/10.1007/s11219-013-9210-6
[24] Ludwig, O., Do, Q. N. T., Smith, C., Cavazza, M., & Moens, M. (2018). Learning to extract action descriptions from narrative text. IEEE Transactions on Games, 10(1), 15–28.
[25] MacDonell, S. G., Min, K., & Connor, A. M. (2014). Autonomous requirements specification processing using natural language processing.
[26] Nazir, F., Butt, W. H., Anwar, M. W., & Khattak, M. A. K. (2017). The applications of natural language processing (NLP) for software requirement engineering-a systematic literature review. In In International Conference on Information Science and Applications (pp. 485–493). Springer.
[27] Nemuraite, L., Skersys, T., Sukys, A., Sinkevicius, E., & Ablonskis, L. (2010). VETIS tool for editing and transforming SBVR business vocabularies and business rules into UML&OCL models. Information Technologies, 21–23.
[28] Nigam, A., Arya, N., Nigam, B., & Jain, D. (2012). Tool for automatic discovery of ambiguity in requirements. International Journal of Computer Science Issues (IJCSI), 9(5), 350–356.
[29] OMG. (2017). Semantics of Business vocabulary and Rules (SBVR). Retrieved from https://www.omg.org/spec/SBVR/About-SBVR/
[30] Osborne, M., & Macnish, C. (1996). Processing natural language software requirement specifications, 229–236.
[31] Rago, A., Marcos, C., & Diaz-Pace, J. A. (2016). Identifying duplicate functionality in textual use cases by aligning semantic actions. Software & Systems Modeling, 15(2), 579–603.
[32] Reshamwala, A., Pawar, P., & Mishra, D. (2013). Review on natural language processing. IRACST Engineering Science and Technology: An International Journal (ESTIJ), 3(1), 113–116.
[33] Robie, M. A. M., Baharom, F., & Mohd, H. (2014). Functional requirements specification for e-tendering system using requirement template. In In Knowledge Management International Conference (KMICe), Langkawi, Malaysia.
[34] Rodrigues, A., Verelst, J., Mannaert, H., Ferreira, D. A., & Huysmans, P. (2014). Towards a system requirements specification template that minimizes combinatorial effects. In In Quality of Information and Communications Technology (QUATIC), 2014 9th International Conference. http://doi.org/10.1109/QUATIC.2014.22
[35] Ryan, K. (1993). The role of natural language in requirements engineering. In In Requirements Engineering, 1993., Proceedings of IEEE International Symposium (pp. 240–242).
[36] Selway, M., Grossmann, G., Mayer, W., & Stumptner, M. (2015). Formalising natural language specifications using a cognitive linguistic/configuration based approach. Information Systems, 54, 191–208.
[37] Shah, U. S., & Jinwala, D. C. (2015). Resolving ambiguities in natural language software requirements: a comprehensive survey. ACM SIGSOFT Software Engineering Notes, 40(5), 1–7. http://doi.org/10.1145/2815021.2815032
[38] Tripathy, A., & Rath, S. K. (2014). Application of natural language processing in object oriented software development. In In Recent Trends in Information Technology (ICRTIT), 2014 International Conference (pp. 1–7).
[39] Umber, A., & Bajwa, I. S. (2011). Minimizing Ambiguity in Natural Language Software Requirements Specification. In ICDIM (pp. 102–107).
[40] Vemuri, S., Chala, S., & Fathi, M. (2017). Automated use case diagram generation from textual user requirement documents. In In Electrical and Computer Engineering (CCECE). (pp. 1–4). IEEE.
[41] Wang, Y. (2016). Automatic semantic analysis of software requirements through machine learning and ontology approach, 21(6), 692–701. http://doi.org/10.1007/s12204-016-1783-3