A Scalable Operational Framework for Requirements Vali- dation Using Semantic and Functional Models

Conference Paper · January 2019

DOI: 10.1145/3305160.3305166

CITATIONS
12

READS
232

1 author:

Issa Atoum
Philadelphia University
35 PUBLICATIONS **407** CITATIONS

SEE PROFILE

A Scalable Operational Framework for Requirements Validation Using Semantic and Functional Models

Issa Atoum
Faculty of Information Technology,
The World Islamic Sciences and Education,
Amman, Jordan
Issa.Atoum@wise.edu.jo

ABSTRACT

A successful operational software depends on adequacy and degrees of freedom in requirements definitions. The software developer in conjunction with the customer validates requirements to ensure the completion of the intended use and the capability of the target application. Notwithstanding, requirements validation is time-consuming, effortless and expensive, and many times involves error-prone manual activities. The difficulty of the problem increases with an increase in the application size, the application domain, and inherit textual requirements constructs. Current approaches to the problem are considered as passive-defect aggregations, domain-specific, or rather fine-grained with formal specifications. We propose a scalable operational framework to learn, predict, and recognize requirements defects using semantic similarity models and the Integration Functional Definition methods. The proposed framework automates the validation process and increases the productivity of software engineers online with customer needs. A proof of concept shows the applicability of our solution to requirements *inconsistency* defects.

CCS Concepts

• Software and its engineering → Software creation and management → Software verification and validation → Process validation • Software and its engineering → Software organization and properties → Software functional properties → Correctness → Consistency

Keywords

ISO 29148; software requirements; requirements validation; semantic similarity; functional models; deep learning.

1. INTRODUCTION

Software requirements artifacts express the customer needs under its associated constraints. Initially, the customers communicate requirements as a statement of needs or objectives documents [1]. At a later stage in the software development, the software engineer transforms them into more detailed software requirement document known as the software requirement specification document (SRS).

The requirement engineering process carries out elicitation, modeling and analysis, management and evaluation, and validation and verification sub-processes [2]; therefore producing a strong customer satisfaction. Several studies showed that the root cause of failures of approximately 37- 70% of projects [3]–[5] are due to inadequate requirements; therefore, the validation process drives the behavior of a successful project. The validation process involves checking up a list of requirements properties or characteristics. The most cited desirable properties in the literature, are the 3C's; *consistency*, *completeness*, and *correctness* [6]. Software engineers consider a set of requirements *inconsistent* if the inclusion of one requirement-element cancels or degrades others. They declare requirements' *completeness* if it assures that the customer has stated all his needs before the project gets validated. If the customer gets the intended results as requested by the customer, requirements are said to be *correct*. These properties and many others are explained in the ISO 29148:2011 model [7]. Therefore, a requirements validation problem solution tackles the recognition of these properties accompanied by the customer objectives.

The most common form of requirements is the textual form [8], usually in a tabular form; each line contains the requirement id, the description, a relative stakeholder's objective, and other related metadata such as the requirement impotence or the risk level. Well-formed textual requirements dramatically reduce the efforts caused by the validation process, usually through language patterns, such as *{"subject," "verb," and "complement"}* or through keywords *{"shall'', "will'', "is''}* that triggers the existence of a requirement defect. Nevertheless, it is hard to have an agreed upon format, or to know even if this will work all the time in large-scale enterprises. Aside from the textual form, a formal specification of the requirements may additionally retain their description in a natural language, but they potentially lose the information perspective on the decision of usefulness to customers. Moreover, the software engineers should generate the formal specification system and choose an appropriate formal system solver. No matter what specification language the developer use, it will miss hidden semantics between requirements such as application usefulness and quality of use [9], [10]. Furthermore, the separation of the crosscutting functional and non-functional properties of formal models is challenging [11]. Even the most large-scale requirements engineering tools (DOORS, Reqtify), prolong little support for traceability [12] and describe the external behavior of requirements in an abstract notion.

The problem under consideration has several solutions in the literature such as surveys, prototypes, and peer-reviews; however, these studies have a limited amount of requirements datasets

ICSIM 2019, January 10–13, 2019, Bali, Indonesia

ACM ISBN 978-1-4503-6642-7/19/01…$15.00
https://doi.org/10.1145/3305160.3305166

[13]–[15]. The promising methods are those that rely on machine learning to predict or learn requirements defects [16]–[19]. Although they are enormous, many works are domain-specific [20][21], fine-grained to limited datasets [22][16], or are at best cases are semi-automated [14]. To the best of our knowledge there is little research that operates on the textual requirement directly using *deep learning* [23], yet, they are intangible. *Deep learning*, a part of machine learning, inspired by human brains, trains the computer by examples on what comes to human logic and it can achieve state-of-the-art accuracy, sometimes exceeding human-level performance.

The object of this work is to capture the requirement *"as is"* in natural language, and then, apply *deep* semantic models to identify requirement defects. Instead of looking for linguistic properties such as part-of-speech, syntactic rules, and patterns, the proposed framework can learn semantic categories (ex. person, company, date.) and their relationships. The framework converts textual requirements to *feature* vectors. Therefore the semantic model finds similarities or relationships between requirements. We argue that, instead of tracing requirements formally, we could find defects at any phase of software development based on textual requirements, the domain datasets, and semantic models.

This paper proposes a functional framework based on IDEF0 (Integration DEFinition) language and a list of techniques from the machine learning community and a set of semantic similarity models. The IDEF0 method is suitable for large enterprise companies when software gets merged or decomposed to plan development activities; therefore, gaining maximum efficiency. The IDEF0 provides a mechanism of communicating functions between developers and the project stakeholders. Figure 1 shows the proposed framework context diagram. Given software requirements (I1), and related domain datasets (I2), using the proposed semantic model (M1) by the project team (M2), a set of recommended actions (O1) is anticipated. The project constraints (C1) and the natural language standards (C2) govern the execution of this IDEF0 function. A project constraint includes laws and regulations, technology barriers, standards and organizations specifications, social and legal responsibilities, market trends, and local culture. The semantic model depends on *features* (main keywords) extracted from textual requirements; therefore, natural language standards such as *accuracy* and *precision* could be used to identify potential requirements conflicts. We illustrate the proposed framework on the requirements of a course management system that allows lecturers to create online content for students and managing courses online [24].

R1: The System shall allow lecturers to limit the number of students subscribing to a course.
R2: The System shall have no maximum limit for the number of course participants ever.
R3: The System shall allow lecturers to specify enrollment policies (based on grade, first come first serve, and department).
R4: The System shall allow lecturers to specify enrollment policies based on grade.

The requirements R1 and R2 are *inconsistent*; both convey similarities except that the second one is the negation of the first. The framework converts the textual requirements to a set of *features*; accordingly, the features turn out to be semantic vectors in the vector space of the framework's adopted datasets. Consequently, requirements similarities are detected using selected semantic similarities techniques; typically, by a cosine similarity. Although R3 an R4 are semantically similar, the first emboss additional *features*; therefore, the *features* can detect the *containment* relationship (part of *traceability*).

The remainder of the paper is structured as follows. Section 2 summarizes related works. In Section 3 we explain the proposed framework. In Section 4, we evaluate the proposed framework, while Section 5, describes the implications and limitations of the proposed framework, and Section 6 provides conclusions and future research.

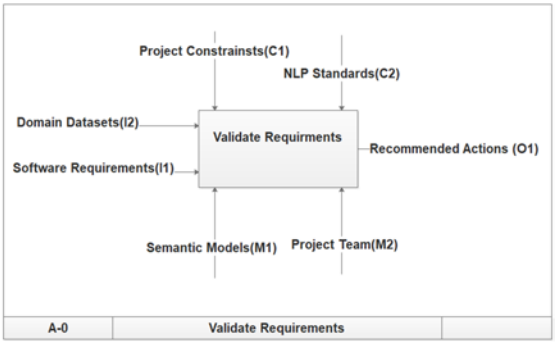

Figure. 1. Context Diagram for the Proposed Framework

2. RELATED WORKS

This section covers the formal and natural language validation approaches.

2.1 Formal Validation Approaches

The QuadREAD project (Quality-Driven Requirements Engineering and Architectural Design) has contributed to a requirement metamodel with formal relationship types based on first-order logic; consequently, the relationship checks *consistency* and *inferencing* properties [24]. The MaramaAIC (Automated Inconsistency Checker) provides automated *traceability* of requirements and visual support to identify and highlight *inconsistency*, *incorrectness*, and *incompleteness* in captured requirements [25]. The model derives an essential use case (EUC) model using textual abstract interactions and pattern library; subsequently, it highlights *inconsistency*; *incompleteness*; *incorrectness* properties. Designed to read like a natural language, the SpeAR (Specification and Analysis of Requirements) captures requirements using the semantics of past LTL(linear temporal logic) [26]; however, engineers will have to take natural language text and determine how to formalize them.

2.2 Machine Learning Approaches

Several machine learning and linguistic models have full usage in requirements elicitation [27], validation and management[28]. Based on an empirical study in a railway system, the SREE (Systemized Requirements Engineering Environment) tool detects *complete*, *clear*, *precise*, *unequivocal*, *verifiable*, *testable*, and *maintainable* properties using patterns [20]. As well as patterns, a set of syntactic rules of natural language constructs is used to detect *non-ambiguity*, specification *completeness*, *consistency*, *understandability* of requirements. The method counts potential defects in each requirement sentence assuming that all sentences have the same weight [29]. Part of the CASE (Scaffolding Scalable Software Services) project[1], the Reqs2Specs module receives

[1]http://www.scasep7.eu

requirements articulated in various modalities and decodes them into system specifications. The Ambiguity Finder of requirements discriminates *ambiguity* based on a set of keywords extracted from requirements, however, the model is seen as a trading *recall* by *precision* [22]. Domain knowledge requirements help software engineers get extra relevant information about an application under consideration using the ELICitation Aid tool (ELICA) [30].

Many natural language approaches are imperfect; they practice their discipline by a parser, and a parts-of-speech identifier to validate requirements [20], leaving language semantics (*deep features)* of software requirements out of scope. Moreover, many tools are domain-specific or property-specific; therefore, traditional machine learning approaches are shallow to detect *traceability* of requirements and are not always accurate. The proposed framework is a constructive *deep learning* approach, similar to ideas in the ORSIM (OpenReq-Similarity) tool [16] and the SenseGraph project [23]. The ORSIM tool process natural language requirement in textual form and estimate *similarity* and *interdependency* with other requirements using a set of tuned natural language processing algorithms. The SenseGraph project [23] applies *deep learning* instead of traditional conventional learning; however, the tool is proprietary and still under development.

2. PROPOSED FRAMEWORK

Figure 2 details the IDEF0 proposed in Figure 1. Under the control of project constraints, privacy concerns of available datasets, using available text processing tools and languages, Function 1 further preprocess the software requirements (I1); therefore, the output of this function is the preprocessed datasets. Acquiring relevant datasets is a tedious process that could include data from search engines, related projects, lesson learned, and any other data that is deemed useful by the data scientist. User requirements, "*as is,*" are not enough for data-hungry semantic models; however, the framework requires additional datasets once. When the proposed framework is operational, the only needed part is the user requirements.

In Figure 2, Function 2 will take these preprocessed compiles of datasets and ensure further preprocessing alongside other related domain datasets (I2) to produce a suitable semantic space. The data scientist systematically accomplishes a set of process to prepare the semantic space that consists of a set of vectors for every item or sentence in the final dataset. This paper will not detail the activities of the data scientist; however, he will be responsible for dropping unneeded data, filtering, tokenization, splitting, selecting machine learning and natural language processing approaches. Moreover, with the project manager, he should also guide the process to annotate datasets if needed and tune selected algorithms for best performance.

Meanwhile, the software project manager with his team will prepare a software requirement quality plan (Function 3) which includes the quality model, the quality measurement metrics and the resource allocations. The plan will determine software requirements properties that need to be validated, assessment of defects, and thresholds that determine if a defect is present. The project constraints, the available quality standards, and natural language processing metrics standards will guide this function. One crucial de facto standard in machine learning and natural language processing is the performance measures: *precision*, *recall,* and *accuracy*. The quality plan in Function 3 uses the *precision* and *recall* as threshold factors that will determine defects existence. The plan also uses the *accuracy* of the model as a guide to the overall performance of the proposed model as compared to the human annotations. The framework stakeholders jointly build much of the activities in this function (Function3); however, in later stages, once the function becomes operational quality metrics are the rule guidelines.

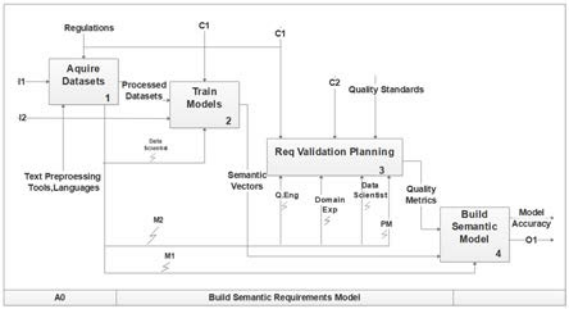

Figure 2. Requirements Validation Semantic Model

The core function, Function 4, will build the semantic model based on prepared quality metrics and two core resources, the data scientist and the customer. The data scientist is responsible for model verification while the customer has the final call to the overall model validation, making sure the output of the semantic model satisfies his needs. In other words, the semantic model will output the *accuracy* or the *precision* of how it is likely that two requirements are *inconsistent*, or if one requirement is *ambiguous*. The semantic vectors carry lots of information about requirements semantics either directly associated with identifiable *features* (ex. *clear* to refer to *ambiguity*) or indirectly by *deep* relationships between textual *features*. Proper processing of this mine will undoubtedly get acceptable results as it did with other domains.

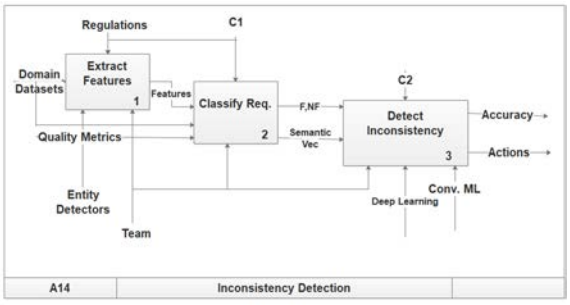

Figure. 3. Proposed Framework Inconsistency detection

The proposed framework hides lots of details from the customer, but at this level, the customer can verify or enable a proper translation of the framework to executable software. Since the proposed framework is supposed to be abstract, we give an example of one software requirements defect *inconsistency* emerged from Function 4 as shown in Figure 3. First, the *feature* extraction extracts the main *features* of each requirement text (ex. time, record, space, graphics). Second, the requirements could be classified into possible targets using selected machine learning algorithms. Therefore, the following functions compare requirements against each other based on their category. The output of this process is the semantic vectors which bring numerical information about software requirements in the target application; Finally, traditional machine learning and *deep learning* methods can find *inconsistencies* in the requirement.

3. EVALUATION AND DISCUSSION

The proposed framework is at early stages in development, and we will implement it using Scrum development; however, it is validated using two techniques. We allow a peer review of the framework with one expert in data science, two senior software engineers, and two customers working on different applications and one senior software project manager. With examples of requirements defects, we explain the framework operations on a piece of paper. Several reviews of the framework continue until we finalize the framework. The problem was not straightforward as the customer did not have any backgrounds about semantic modeling; therefore, we reach a middle point that makes the model operational.

We run a prototype on the two requirements discussed earlier[24], R1 and R2 shown in column 1 of Table 1. The second column shows the list of potential *features* (one and two grams) extracted from the requirement. Typically, the framework converts the *features* to semantic vectors; however, we are not reporting the semantic vectors, a list of probabilities of more than 300 entries for each word in the requirements vocabulary. Accordingly, Table 1 classifies the two requirements as functional. This classification could be detected using an algorithm such as Naïve Bayes or SVM.

The proposed systematic framework shall be able to detect the quality characteristic of the requirement shown in the fourth column. Therefore, the proposed framework predicts that first and the second requirement are *unambiguous* because no *features* such as *{may, could, not clear}* is available in the requirements. Moreover, the framework shall detect defect between more than one requirement; consequently, the two requirements are *inconsistent* as the second has potential affectable *features {maximum limit}* compared to *{limit number}* in the first requirement.

The customer and the data scientist examined the example, and both agreed on this initial finding. We draw attention to the reader that the essential part of this framework is the customer who should agree on the approach, while the technical team implements the framework in details [31]. According to natural language processing communities, once the developer has enough data and a good semantic model, the results become a foregone conclusion [10], [32]–[34].

Table 1: An example of the proposed framework.

Req	Features	Classification	Detected property
R1	{system, allow, lecture, limit, number, student, subscribe, course, limit number} {system allow, allow lectures, limit number, number student, student subscribe, subscribe course}	Functional	Necessary Unambiguous Feasible Singular Complete
R2	{system, have, maximum, limit, number, course, participant} {system have, have maximum, maximum limit, limit number, number course, course participant}	Functional	Necessary Unambiguous Feasible Singular Complete

4. LIMITATIONS AND IMPLICATIONS

Although the proposed framework is essential in building an application that can discover defects in requirements at any stage of software development, it has several limits. We did not fully implement the framework; it is a prototype; therefore, several issues may come across the development. The most troublesome part of the natural language processing algorithms is the availability of datasets [35]. The literature reported many datasets; however, many of them are domain-specific or not disclosed due to privacy concerns. As datasets semantics increase, the proposed framework becomes more accurate. At later stages, the framework should support a way for customers to upload their datasets without disclosing or conflicting any privacy concerns.

This paper identifies implications to current software requirements defects models. Some properties of software quality could be challenging to quantify. For example, the *"necessary"* property is not discussed in studied literature and can not be easily quantified. This property could be detected given a GUI prototype to the user but may be hard to be detected before requirements get detailed. The same could be deduced to the *"implementation free"* property. During the early stages of requirements elicitation process, it is not sure what software artifacts could be developed and whether a requirement will reuse or rebuild existing software components. The *"bounded"* property is related to the target application scope creep that may be identified at later development stages through the change request process, lessons learned from similar projects and the history of change requests. Traditional machine learning algorithms may support this property by counting the number of changes requests related to specific requirement item and thus when a threshold has reached a defect is detected. Therefore, the implications of the proposed framework to requirements engineering are to unify definitions of requirements engineering defect properties and to link the requirements defects to the software development lifecycle type. Junking requirements in phases of Scrum, Kanban or XP models influence the interrelationship between requirements and its associated potential defects.

5. CONCLUSION

Several requirements defect models could be classified as concrete formal models, shallow machine learning models, or abstract deep learning models. One main issue is that models tend to be incomplete, semi-automated, or domain specific. The proposed framework guides the implementation of a new scalable and systematic framework based on the IDEF and a proposed semantic model. The critical contribution of the framework is that it is independent of the software domain, textual requirements' language; therefore, the framework suggests building a complete software requirement defect model. A set of selected stakeholders of customers and data scientists guided a systematic development of the proposed framework. They found that the framework is deemed applicable under certain restrictions. The primary re-

striction is the limited available set of datasets. In the future, we will collect as many datasets as possible and prepare a complete prototype of the proposed framework.

6. REFERENCES

[1] ISO/IEC/IEEE, "IEEE 29148-2011 - ISO/IEC/IEEE International Standard - Systems and software engineering -- Life cycle processes --Requirements engineering," 2011.

[2] J. Dick, E. Hull, and K. Jackson, "A Generic Process for Requirements Engineering," in Requirements Engineering, Cham: Springer International Publishing, 2017, pp. 33–56.

[3] C. Lindquist, "Required: Fixing the requirements mess: The requirements process, literally, deciding what should be included in software, is destroying projects in ways that aren't evident until it's too late. Some CIOs are stepping in to rewrite the rules," CIO, vol. 19, no. 4, p. 1, 2005.

[4] A. Mandal and S. C. Pal, "Identifying the reasons for software project failure and some of their proposed remedial through BRIDGE process models," Int. J. Comput. Sci. Eng., vol. 3, no. 1, pp. 118–126, 2015.

[5] E. Larson, "still don't have time to manage requirements: My project is later than ever," in PMI® Global Congress, 2014.

[6] D. Zowghi and V. Gervasi, "The Three Cs of Requirements: Consistency, Completeness, and Correctness," Proc. 8th Int. Work. Requir. Eng. Found. Softw. Qual., no. March, pp. 155–164, 2002.

[7] Q. Requirements and I. E. C. ISO, "IEEE. 29148: 2011-Systems and software engineering-Requirements engineering," 2011.

[8] M. Kassab, C. Neill, and P. Laplante, "State of practice in requirements engineering: contemporary data," Innov. Syst. Softw. Eng., vol. 10, no. 4, pp. 235–241, 2014.

[9] I. Atoum and A. Otoom, "Mining Software Quality from Software Reviews : Research Trends and Open Issues," International Journal of Computer Trends and Technology (IJCTT), vol. 31, no. 2. Seventh Sense Research GroupTM, pp. 74–83, 2016.

[10] I. Atoum, "A Novel Framework for Measuring Software Quality-in-use based on Semantic Similarity and Sentiment Analysis of Software Reviews," J. King Saud Univ. - Comput. Inf. Sci., p. , 2018.

[11] A. Rashid, A. Moreira, and J. Araújo, "Modularisation and Composition of Aspectual Requirements," in Proceedings of the 2Nd International Conference on Aspect-oriented Software Development, 2003, pp. 11–20.

[12] F. R. Golra, F. Dagnat, J. Souquières, I. Sayar, and S. Guerin, "Bridging the Gap Between Informal Requirements and Formal Specifications Using Model Federation," in Software Engineering and Formal Methods, 2018, pp. 54–69.

[13] L. Hua-Xiao, W. Shou-Yan, and J. Ying, "A Tool to Verify the Consistency of Requirements Concern Model," in 2013 International Conference on Computational and Information Sciences, 2013, pp. 1955–1958.

[14] M. Kamalrudin, J. Hosking, and J. Grundy, "MaramaAIC: tool support for consistency management and validation of requirements," Autom. Softw. Eng., vol. 24, no. 1, pp. 1–45, 2017.

[15] Y. Elrakaiby, A. Ferrari, P. Spoletini, S. Gnesi, and B. Nuseibeh, "Using Argumentation to Explain Ambiguity in Requirements Elicitation Interviews," in 2017 IEEE 25th International Requirements Engineering Conference (RE), 2017, pp. 51–60.

[16] C. A. Nari, C. Palomares, and X. Franch, "ORSIM: Integrating existing software components to detect similar natural language requirements," in 24th International Working Conference of Software Eningeering: Foundation for Software Quality, 2018.

[17] D. M. Berry, A. Ferrari, and S. Gnesi, "Assessing tools for defect detection in natural language requirements: Recall vs precision," 2017.

[18] M. Narouei, H. Takabi, and R. Nielsen, "Automatic Extraction of Access Control Policies from Natural Language Documents," IEEE Trans. Dependable Secur. Comput., vol. 5971, no. c, pp. 1–13, 2018.

[19] T. Diamantopoulos, M. Roth, A. Symeonidis, and E. Klein, "Software requirements as an application domain for natural language processing," Lang. Resour. Eval., vol. 51, no. 2, pp. 495–524, 2017.

[20] A. Ferrari et al., "Detecting requirements defects with NLP patterns: an industrial experience in the railway domain," Empir. Softw. Eng., pp. 1–50, 2018.

[21] H. Guo, Ö. Kafal?, and M. P. Singh, "Extraction and Formal Representation of Natural Language Requirements from Breach Reports," in Fifth International Workshop on Artificial Intelligence for Requirements Engineering, 2018.

[22] S. F. Tjong and D. M. Berry, "The design of SREE - A prototype potential ambiguity finder for requirements specifications and lessons learned," Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 7830 LNCS, pp. 80–95, 2013.

[23] R. Garigliano, D. Perini, and L. Mich, "Which semantics for requirements engineering: From shallow to deep," CEUR Workshop Proc., vol. 2075, no. March, 2018.

[24] A. Goknil, I. Kurtev, K. van den Berg, and J.-W. Veldhuis, "Semantics of trace relations in requirements models for consistency checking and inferencing," Softw. Syst. Model., vol. 10, no. 1, pp. 31–54, 2011.

[25] M. Kamalrudin, J. Hosking, and J. Grundy, MaramaAIC: tool support for consistency management and validation of requirements, vol. 24, no. 1. Springer US, 2017.

[26] A. W. Fifarek, L. G. Wagner, J. A. Hoffman, B. D. Rodes, M. A. Aiello, and J. A. Davis, "SpeAR v2.0: Formalized past LTL specification and analysis of requirements," Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 10227 LNCS, pp. 420–426, 2017.

[27] I. Atoum, "Requirements Elicitation Approach for Cyber Security Systems," i-manager's J. Softw. Eng., vol. 10, no. 3, pp. 1–5, 2016.

[28] A. Vision, A. Ferrari, F. D. Orletta, A. Esuli, and S. Gnesi, "Natural Language Requirements Processing : A 4D Vision," IEEE Softw., 2017.

[29] F. Fabbrini, M. Fusani, S. Gnesi, and G. Lami, “The linguistic approach to the natural language requirements quality: benefit of the use of an automatic tool,” Proc. 26th Annu. Softw. Eng. Work. 2001, pp. 97–105, 2001.

[30] Z. S. H. Abad, V. Gervasi, D. Zowghi, and K. Barker, “ELICA: An Automated Tool for Dynamic Extraction of Requirements Relevant Information,” in Fifth International Workshop on Artificial Intelligence for Requirements Engineering Requirements Engineering (AIRE’18), 2018.

[31] J. T. Wu et al., “Behind the scenes: A medical natural language processing project,” Int. J. Med. Inform., vol. 112, pp. 68–73, 2018.

[32] R. Collobert and J. Weston, “A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning,” in Proceedings of the 25th International Conference on Machine Learning, 2008, pp. 160–167.

[33] I. Atoum and A. Otoom, “Efficient Hybrid Semantic Text Similarity using Wordnet and a Corpus,” International Journal of Advanced Computer Science and Applications(IJACSA), vol. 7, no. 9. The Science and Information (SAI) Organization Limited, pp. 124–130, 2016.

[34] I. Atoum, C. H. Bong, and N. Kulathuramaiyer, “Towards Resolving Software Quality-in-Use Measurement Challenges,” Journal of Emerging Trends in Computing and Information Sciences, vol. 5, no. 11. pp. 877–885, 2014.

[35] W. Zogaan, P. Sharma, M. Mirahkorli, and V. Arnaoudova, “Datasets from Fifteen Years of Automated Requirements Traceability Research: Current State, Characteristics, and Quality,” Proc. - 2017 IEEE 25th Int. Requir. Eng. Conf. RE 2017, pp. 110–121, 2017.