

Specification of Non-Functional Requirements: A Hybrid Approach

Unnati S. Shah¹, Sankita J. Patel², Devesh C. Jinwala²

¹ C. K. Pithwalla College of Engineering and Technology, Surat, India
unnati.shah25@gmail.com

²S. V. National Institute of Technology, Surat, India
{sankitapatel, dcjinwala}@gmail.com

Abstract. When specifying user requirements, not only it is critical to ensure correct and unambiguous specification of functional requirements, but also that of non-functional requirements (NFRs). In fact, resolving ambiguities from user specified natural language NFRs and specifying the correct ones in a formal language have attracted significant attention. Our current research focuses on the issues pertaining the same. We observe that it is a usual practice for a user to narrate the NFRs in natural language and the requirement engineers manually try to express the same, using some semi-formal or formal language notations. However, inaccurate and the laborious manual approach may fail to detect all the NFRs and correctly remove the ambiguities in those detected. Hence, current research attempts have focused on automating the conversion of natural language NFRs to formal notations.

In literature, there exist numerous approaches that take requirements as input and output the extended UML counterpart including NFRs. However, majority of the approaches do not support ambiguity resolution and verification of the extracted NFRs that are fairly essential. In this paper, we propose and discuss a hybrid approach viz. NFRs-Specifier, that attempts to resolve ambiguities, extract NFR's, perform verification and generate NFRs specification by means of the extended UML model.

Keywords: Requirements Engineering; Ambiguity; Natural Language Processing; Non-functional Requirements; Requirements Classification; Unified Modeling Language; Ontology

1 Introduction

Requirements Engineering (RE) is one of the most vital activities in the entire software development life cycle [1]. The RE activity often starts with the vaguely defined requirements [2, 3] that results eventually in a software requirements specification document. To make the RE process effective there exists various RE approaches viz. View-point Oriented RE (VORE) [4], Aspect Oriented RE (AORE) [5], Goal Oriented RE (GORE) [6-9] and Ontology based RE (ORE) [10] among

others. The success of any software depends on correct and unambiguous specification of Functional Requirements (FRs) and Non-Functional Requirements (NFRs) [11]. The FRs are relatively easy to identify and specify. However, to capture and specify the NFRs is difficult, as compared. This is so, because often the users' narration of the NFRs is vague and is hidden in the FRs. Thus, a major problem in RE is identification of unrevealed NFRs, conflict resolution and their unambiguous specification.

The NFRs [12-15] are also known by a relatively colloquial term viz. Quality Requirements (QR). To present the QR, there exist four basic quality models (viz. Boem [16], McCall [17], FRUPS [18] and Dromey [19]) that provide quality attributes in the hierarchy.

As per our literature survey, numerous approaches exist, that deal with NFRs activities viz. elicitation, classification, verification and specification. Typically, users and requirements engineers informally (manually) identify the NFRs from the requirements documents using their experience and expertise [20-24] or use a formal setup e.g. the NFR Framework [25]. The NFR Framework is a Goal Oriented Approach (GOA), used to represent NFRs graphically by means of a soft-goal interdependency graph without referring a quality model.

Many researchers [26-29] have based their work on the NFR Framework and have treated NFRs as a soft-goal and FRs as a hard goal. However, it is important to the following counter-view [30, 31]:

1. The NFRs/FRs could be treated as hard-goal as well as soft-goal
2. There exists no clear cut boundary between NFRs and FRs and
3. It is difficult to integrate NFRs (the graphical notations) with the FRs (UML Models)

In support of the claims 1 and 2, authors [32] provide precise definitions of Soft Goal, Quality Goal and Quality Constraints. To deal with the integration issue (claim-3), in [24, 33-35] NFRs are integrated by means of extending UML models (viz. use-case, class, activity, sequence, etc). However, this approach models certain NFRs due to limited knowledge regarding quality focus expected.

We reiterate that it is error-prone to analyze a large set of software requirements and identify relationships amongst them - especially when using inaccurate, time consuming and laborious manual approach is employed. Obviously, expecting a user to specify the requirements in a formal language that is cryptic and that requires sophisticated skills is idealistic.

On the other hand, semi-formal approach (viz. natural language processing, machine learning, etc) helps to reduce human efforts in identifying and classifying NFRs from requirements documents accurately. The machine learning approaches viz. supervised/ semi-supervised and unsupervised provide an ease by classifying requirements without human expert. As compared to informal approaches, this approach is cheap, flexible and less labor intensive.

However, it is usually impossible to achieve accuracy and high performance without a lot of training labeled data set. Again, the manual annotation process is time consuming and error prone. Moreover, changes are inevitable in real world and hence when there are changes in the domain, words/terms used in the requirements or the writing style, there is an imminent need to retrain the machine [36]. The Natural Language Processing (NLP) helps to extract NFRs from requirements documents [37, 38]. However, NLP cannot help to provide additional information regarding application domain. The NFRs are difficult to describe completely and precisely due to vague, conceptual, and the subjective nature [37]. Furthermore, it is unreasonable to ask users to provide their NFRs explicitly because they are related to specific domains and affected by context. In addition, it is difficult to meet the changing needs of the environment and to describe them in unified and standardized form. To identify NFRs definitely there is a need to provide the domain knowledge support at the time of interview. Building the ontology based on domain knowledge and quality models gives a formal, explicit specification of a shared conceptualization [39]. It helps domain users- to suggest their requirements effectively and requirements analysts- to understand and model the requirements accurately. The ontology can promote common understanding of NFRs among developers, and can be used as a basis for specifying NFRs. In literature, there exists number of ontology based approaches [40-49] to specify NFRs.

After analyzing existing approaches/tools, we classify them in three categories viz. formal, informal and semi-formal. We observe that these approaches (viz. informal, semi-formal and formal) are not competitive, instead complementary. Furthermore, we observe that the informal approach mainly focuses on elicitation and specification of the NFRs. The semi-formal approach, focus on the classification of the NFRs and the formal methods (viz. Z notations [50], UML-B [51]) help for formal specification of the NFRs.

Our proposed approach viz. NFR-Specifier, mainly focuses on extracting all possible NFRs from requirements documents and provides specification after resolving ambiguity. To achieve this, we use a hybrid approach – combination of NLP, machine learning and ontology.

2 PROPOSED APPROACH

We propose a semi-automated approach called NFR-Specifier, aims to generate accurate specification from informal requirements including NFRs as shown in figure 1. The approach consists of five modules viz. preprocessing, ambiguity resolving, SRS ontology formation, UML diagram generation and NFRs classification. Initially, requirements engineer gathers domain knowledge from users by means of various communication approaches viz. questionnaires, interviews, checklist, prototyping, meetings, among others. Once the communication phase is over, the requirements engineer represents the collected information by means of a text files, documents, graphs or UML models (viz. use-case, class, sequence diagram). These initial requirements are ambiguous in nature.

The **preprocessing module** takes input as natural language requirements and produces normalized natural language requirements. The module performs three tasks viz. sentence splitting, Part-Of-Speech (POS) tagging and normalizing. It performs syntactic reconstruction to split a complex sentence into simple sentences to extract all possible information from the requirements document. We use the Stanford parser, for lexical-syntactic analysis and WordNet [52] to determine context knowledge. Each token is analyzed and classified into its respective POS (Part-Of-Speech) classification viz. noun, verb, pronoun, adverb, helping-verb, adjective, prepositions, etc. Furthermore, we use dependencies [53] (binary relations that give a grammatical relation between a head and a dependent relative in a sentence) generated by the Stanford parser to identify the semantic relationships between words. The normalizing process also performs spelling and grammar checking. We perform a comparative analysis of ambiguity (having more than one meaning to a word/sentence) resolving approaches/tools [3]. The analysis shows that resolving ambiguity at an early stage makes initial requirements clear, complete and precise. Furthermore, not all rather, ambiguities viz. Anaphora, attachment, event anaphora, coordination, among others affects the SRS.

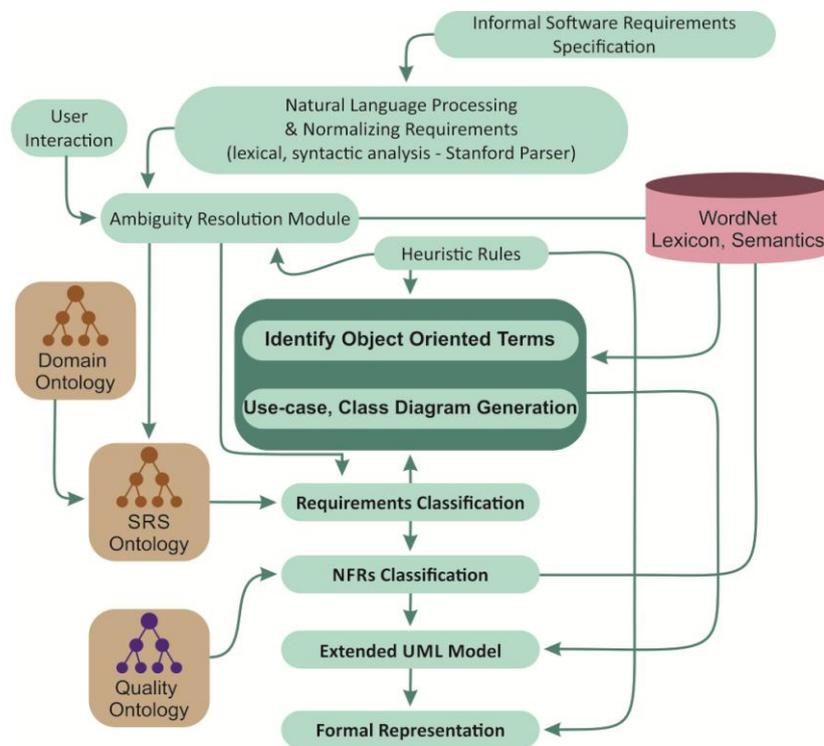


Fig. 1. Architecture of Proposed Approach

The **ambiguity resolving module** takes the input as a normalized requirement and produces the unambiguous natural language requirements. The module identifies

ambiguous requirements and suggests the most suitable solution to resolve the ambiguity. We propose an architecture viz. ARUgen that is aimed to resolve ambiguities from informal requirements [54]. Our tool ARUgen mainly deals with ambiguities viz. pronoun anaphora, verb anaphora and coordination. After resolving ambiguity, we generate Software Requirements Specification (SRS) Ontology semi-automatically with the help of pre-build domain ontology and rule based approach. The SRS ontology helps to identify in-depth and complete knowledge of the application requirements. After developing SRS ontology, the next aim is to generate UML models (semi-) automatically. We extract Object Oriented Terms (OOT) viz. subject/class, object/class, attributes, methods using rule based approach and Stanford dependencies as shown in table 1.

Table 1. Stanford Dependencies used to Extract OOT

OOT	Stanford Dependencies
Subject/	csubjpass (clausal passive subject); nsubj (nominal subject)
Class	nsubjpass (passive nominal subject); xsubj (controlling subject);
Object/	doj (direct object); iobj (indirect object), pobj (object of a
Class	preposition);
Attribute	acomp; advmod (adverbial modifier); amod (adjectival modifier), (String/number) (numeric modifier), npadvmod: noun phrase as adverbial modifier;
Method	aux: auxiliary; auxpass: passive auxiliary, complm: complementizer, rcmmod: relative clause modifier; xcomp: openclausal complement).

Relationships between classes are extracted using the rules listed in table 2.

Table 2. Rules To Identify Relationship Between Classes

Relationship	Rules
Association	(Noun + Verb + Noun) or Noun + Keyword + Noun Keyword: has, next to, works for, contained in, talk to
Inheritance	(Subject + Keyword + Object) Keyword: maybe/ is type of
Composition	(Subject + Keyword + Object) Keyword: comprises, have, include, possess, contains
Aggregation	Subject+Phrase + Object Phrase: is a part of
Cardinality	Stanford dependency: predet-(predeterminer) Predet (the, a, proper noun) -> 1 Predet (all, many, more)-> *

The extracted entities are used to generate UML models automatically. The approach provides flexibility to modify the auto-generated UML models. After that, we perform

requirements clustering based on noun. Here, we use hierarchical clustering [55]. In literature, we have identified various promising machine learning algorithms used for requirements classification [56-58]. We observe that the accuracy of machine learning algorithm depends on various factors such as feature selection, distance/similarity measures, dataset, among others [55, 57]. We provide flexibility to adjust classification parameters and to add/update/delete irrelevant requirement classes. Once we have functional grouping of the requirements based on nouns, we verify these requirements with the system generated use-case model as shown in fig. 1. The approach then extracts possible NFRs using pre-developed quality ontology based on specific parts of standard quality models using protégé tool. We apply classification to identify NFRs related requirements. The module takes normalized unambiguous requirements as an input and produces the NFRs classification using ontology and machine learning approach. The extracted NFRs are integrated with UML models viz. use-case diagram. We provide a generalized algorithm in figure 2 and a brief summary of each module in table 3.

- | |
|--|
| <ul style="list-style-type: none"> A. Initial informal requirements B. Apply POS Tagging //Natural language processing C. Resolve coordination ambiguity (and, or, as well as, but not) D. Resolve Anaphora Ambiguity //Rule based <ul style="list-style-type: none"> a. Identify Anaphora ambiguity b. Identify Antecedent c. Avoid Non-anaphoric anaphora d. Generate suitable antecedent for anaphora//user input E. Create Software Requirements Specification Ontology <ul style="list-style-type: none"> a. Extract nouns from the requirements b. Group similar nouns based semantics and give generic name to them and create a hierarchy F. Apply requirements clustering//Machine learning <ul style="list-style-type: none"> a. Apply stop word removing and stemming b. Select feature (viz. noun, verb, adjective)//user input c. Generate distance/similarity measure matrix d. Group similar requirements in to a cluster G. Create UML Diagram <ul style="list-style-type: none"> a. Extract Class, Attributes and Methods b. Extract relationship among Classes c. Extract cardinalities d. Extract class diagram e. Extract actors and use-cases// //using Stanford Dependencies f. Generate Use-case Diagram H. NFRs Classification//using Ontology <ul style="list-style-type: none"> a. Identify NFRs using Quality Ontology b. Refine NFRs related requirements I. Integrate NFRs to the UML diagram |
|--|

Fig. 2. Procedure: NFRs-Specifier

Table 3. A Brief Summary Of Each Module Of The Proposed Approach

Module	Task	Technology/ Approach	Output
Preprocessing	Sentence Splitting, POS tagging, Normalizing		Normalized Requirements
Ambiguity Resolving	Anaphora ambiguity (viz. event, verb, personal pronoun, possessive pronouns, Wh-pronoun, Wh-adverb), Coordination ambiguity and Attachment ambiguity	-Stanford Parse -WordNet -NLP -Heuristic Rules	Unambiguous Requirements
Create SRS Ontology	Based on prebuild domain ontology create a software requirements specification ontology	-Rule based Approach -Protégé tool	SRS Ontology
Create UML Models	-Identify OOT -Extract Relationship -Extract subject, object and dependency agent to identify possible actors -Extract verb phrases to identify possible use-cases -Stemming words	-Rule based Approach -Stanford Dependencies -WordNet	Use-case Model
Requirements Classification	-Extract nouns -Apply distance/similarity measures -Apply clustering algorithm	-Machine learning Approach -Hierarchical Clustering -Hamming Distance	Requirements Clusters
NFRs Classification	-Extract NFRs -Classify NFRs	-Ontology based Approach -Keyword search -Rule based Approach	Extracted NFRs
Extended UML Models	-Integrating NFRs in UML models	-Rule based approach	Extended Use-case Model

3 A CASE STUDY

In this section, we provide the detailed analysis of the proposed approach (figure 2) on Automated Teller Machine (ATM) as a case study.

A. Initial requirements

- R1: The ATM interacts with the customer to gather transaction information.
- R2: The bank computer gets the transaction information from the ATM to verify an account **and** to process a transaction.
- R3: Each bank may be processing transactions from several ATMs at the same time.
- R4: The customer interacts with the ATM network via the ATM.
- R5: **It** must be very easy for **them** to use the ATM.
- R6: The ATM network has to be available 24 hours a day.
- R7: The ATM network should provide maximal security.

B. POS Tagging

- R1: The|DT ATM|NNP interacts|VBZ with|IN the|DT customer|NN to|TO gather|VB **transaction|NN information|NN** .|.
- R2: The|DT **bank|NN computer|NN** gets|NNS the|DT **transaction|NN information|NN** from|IN the|DT ATM|NNP to|TO verify|VB an|DT account|NN and|CC to|TO **process|NN** a|DT transaction|NN .|.
- R3: Each|DT bank|NN may|MD be|VB processing|VBG transactions|NNS from|IN **several|JJ ATMs|NNS** at|IN the|DT **same|JJ time|NN** (: performance|NN): .|.
- R4: The|DT customer|NN interacts|VBZ with|IN the|DT **ATM|NNP network|NN** via|IN the|DT ATM|NNP .|.
- R5: **It|PRP** must|MD be|VB **very|RB easy|JJ for|IN them|PRP** to|TO use|VB the|DT ATM|NNP (:|NNP usability|NN): .|.
- R6: The|DT **ATM|NNP network|NN** has|VBZ to|TO be|VB **available|JJ 24|CD hours|NNS** a|DT day|NN (: availability|NN): .|.
- R7: The|DT ATM|NNP network|NN should|MD provide|VB **maximal|JJ security|NN** (: security|NN)

C. Resolve coordination Ambiguity

The approach identifies coordination ambiguity in requirement R2. It will split the requirement and make two separate requirements as:

R2.1: The **bank_computer** gets the **transaction_information** from the ATM to verify an account.

R2.2: The **bank_computer** gets the **transaction_information** from the ATM to process a transaction.

D. Resolve Anaphora Ambiguity

The approach identifies anaphora ambiguity in requirement R5 as the requirement R5 contains the keyword “it” and “them”. The system resolve the ambiguity “them” automatically and rewrite the requirement R5 as

R5: **It** must be very easy for **customer** to use the ATM.

E. Create SRS Ontology

To generate the SRS ontology automatically we have used the pre-developed domain ontology as shown in figure 3. The partial output of the generated SRS ontology is shown in figure 4.

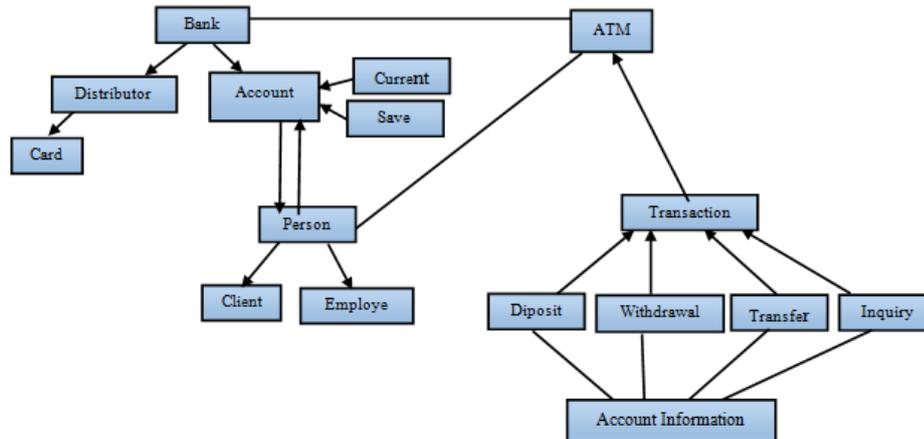


Fig. 3. Domain Ontology

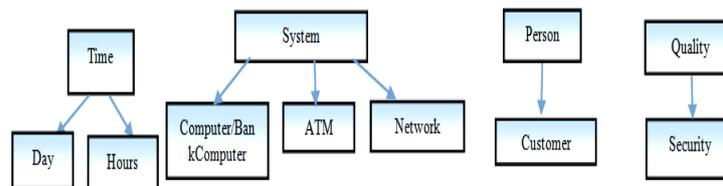


Fig. 4. SRS Ontology

F. Requirements Clustering

We perform requirements clustering based on noun and using hamming code distance measure. We extract nouns (viz. ATM, customer, transaction_Information, Bank_computer, account, transaction, bank, time, ATM_network, hours, day, security) from R1 - R8 to calculate the distance of two requirements. Table 4 shows the distance matrix calculation for the requirements R1 to R8.

G. Extracted Nouns, Verbs and Adjectives to create UML Diagram

Noun: ATM, Customer, Transaction_Information, Bank_Computer, Account, Transaction, Bank, Time, ATM_Network, Hours, Day, Security

Verb: Interacts, get, gather, verify, process, use, provide

Adverb/Vauge (adverb+...+noun): Several ATMs, same time, very easy for user, available 24 hours, maximal security

Table 4. Distance matrix

	R1	R2	R3	R4	R5	R6	R7	R8
R1	-	3	3	4	2	1	6	5
R2	3	-	2	5	5	4	7	6
R3	3	2	-	5	5	4	7	6
R4	4	5	5	-	4	3	6	5
R5	2	5	5	4	-	1	4	3
R6	1	4	4	3	1	-	5	4
R7	6	7	7	6	4	5	-	3
R8	5	6	6	5	3	4	3	-

We apply the hierarchical clustering and get the cluster as follows:

1. Cluster1 : R1, R2, R3
2. Cluster2 : R4, R6
3. Cluster3 : R5, R7, R8

Once the requirements are clustered, we verify these requirements with the generated use-case model as shown in figure 5.

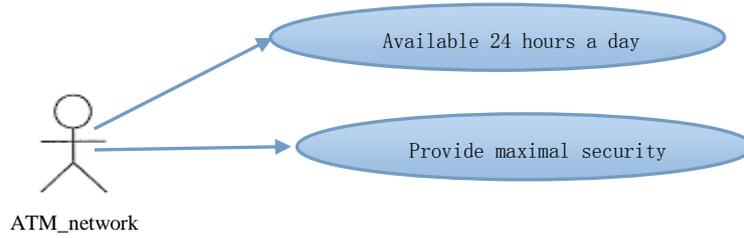


Fig. 5. Use-Case Model

H. NFRs Classification

Using pre-developed quality ontology based on standard quality model, we extract the NFRs from the requirements viz. security, availability, performance and the relationship to the requirements as shown in table 5.

Table 5. Requirements and related NFRs

Requirement	NFRs
R4	Performance

	Execution time Response Time
R6	Usability
R7	Availability
R8	Security Authenticity Integrity Availability

I. Integrate NFRs to the UML Diagram

Finally, the extended use-case model generated as shown in figure 6.

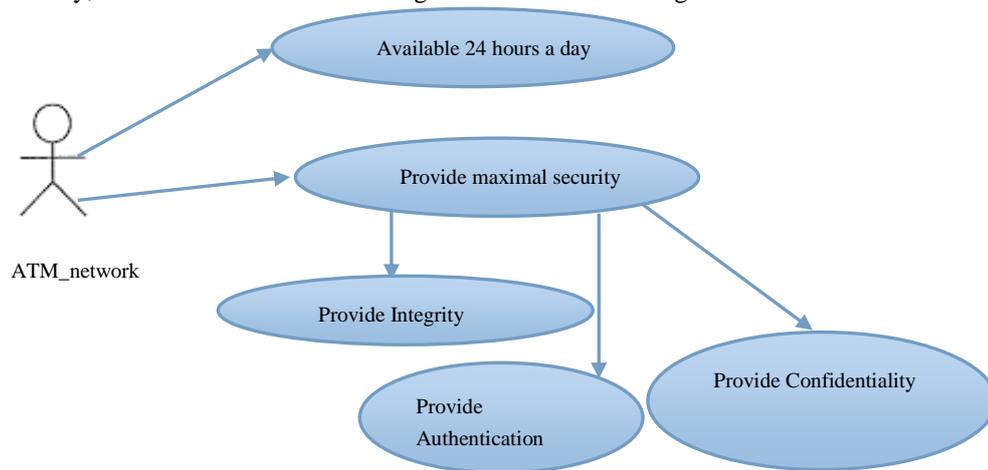


Fig. 6. Extended Use-Case Model

4 DISCUSSION

The presented approach seems easy if human intelligence applies, but to make the process automated, requires a lot of training dataset and domain knowledge. Furthermore, the machine learning algorithms may not perform well if NFRs occur that are not relevant to the predefined category. In order to adapt the change in the application domain, the algorithms need to be re-trained and re-evaluated that again require manual efforts. On the other hand, to provide a complete and precise NFRs specification, we need to identify all possible NFRs conflicts as it may happen that one NFR affect (positively or negatively) other NFRs. To provide the conceptualization, it is required to identify the dependencies exist between NFRs. The NFRs provides constraints on FRs, thus the change in NFRs may cause the change in FRs. Though, we investigate the positive impact of combining rule based and machine learning approaches on classification of NFRs using quality ontology, we need to investigate the impact of hybrid clustering algorithms for classification of NFRs.

5 CONCLUSIONS

The correct and precise software requirements specification is required for the success of the software. If the requirements are not extracted and analyzed using an engineering approach, the errors that creep into the software are detected in the later stages of the software development, leads to higher costs for changes. In this paper, we present a hybrid approach that provides a specification of NFRs. It addresses the problems viz. normalizing, ambiguity resolving, requirements clustering, NFRs classification and verification using efficient natural language processing, a set of rules, ontology and machine learning approaches. After analyzing feasibility of the approach of case study, we conclude that the deployment of the approach in the RE practice would have a positive impact.

References

1. Sommerville, Ian, and Pete Sawyer, Requirements engineering: a good practice guide, John Wiley & Sons, Inc., 1997.
2. Nuseibeh, Bashar, and Steve Easterbrook, "Requirements engineering: a roadmap," Proceedings of the Conference on the Future of Software Engineering, ACM, 2000.
3. Shah US, Jinwala DC, "Resolving Ambiguities in Natural Language Software Requirements: A Comprehensive Survey," ACM SIGSOFT Software Engineering Notes, vol. 40(5), pp. 1-7, 2015.
4. Pu, YuNing, and Qiang Liu, "A Viewpoint-Oriented Requirements Elicitation Integrated with Aspects," Computer Science and Information Engineering, WRI World Congress On, Vol. 7, IEEE, 2009.
5. Rashid, A., and Chitchyan, R, "Aspect-oriented requirements engineering: a roadmap," In Proceedings of the 13th international workshop on Early Aspects, ACM, pp. 35-41, 2006.
6. Anwer, S., and Ikram, N., "Goal oriented requirement engineering: A critical study of techniques," In Software Engineering Conference, APSEC 2006. 13th Asia Pacific, IEEE, pp. 121-130, 2006.
7. Van Lamsweerde, A. and Letier, E., "Handling obstacles in goal-oriented requirements engineering," Software Engineering, IEEE Transactions on, vol. 26 (10), pp.978-1005, 2000.
8. Kavakli, E., "Goal-oriented requirements engineering: A unifying framework," Requirements Engineering, vol. 6(4), pp.237-251, 2002.
9. Lapouchnian, A., "Goal-oriented requirements engineering: An overview of the current research," University of Toronto, p.32, 2005.
10. Lee, Seok Won, and Robin Gandhi, "Ontology-based active requirements engineering framework," In Software Engineering Conference, APSEC'05. 12th Asia-Pacific, IEEE, pp. 8-pp, 2005.

11. IEEE Computer Society. Software Engineering Standards Committee, and IEEE-SA Standards Board. "IEEE Recommended Practice for Software Requirements Specifications." Institute of Electrical and Electronics Engineers, 1998.
12. Davis, Alan M. Software requirements: objects, functions, and states. Prentice-Hall, Inc., 1993.
13. Anton, Ana I, "Goal identification and refinement in the specification of software-based information systems," (1997).
14. Jacobson, Ivar, Grady Booch, James Rumbaugh, James Rumbaugh, and Grady Booch, "The unified software development process," Vol. 1, Reading: Addison-wesley, 1999.
15. Chung, Lawrence, Brian A. Nixon, Eric Yu, and John Mylopoulos, "Non-functional requirements in software engineering," Springer Science & Business Media, Vol. 5, 2012.
16. Boehm, Barry W., John R. Brown, and Mlity Lipow, "Quantitative evaluation of software quality," In Proceedings of the 2nd international conference on Software engineering, IEEE Computer Society Press, pp. 592-605, 1976.
17. McCall, J. A., P. K. Richards, and G. F. Walters. Factors in Software Quality, Volumes I, II, and III, US Rome Air Development Center Reports NTIS AD/A-049 014. NTIS AD/A-049 015 and NTIS AD/A-049 016, US Department of Commerce, 1977.
18. Grady, Robert B., and Deborah L. Caswell, "Software metrics: establishing a company-wide program," (1987).
19. Dromey, G.R., "A model for software product quality," Software Engineering, IEEE Transactions on, vol. 21(2), pp.146-162, 1995.
20. Alsaleh, S. and Haron, H., The Most Important Functional and Non-Functional Requirements of Knowledge Sharing System at Public Academic Institutions: A Case Study," Lecture Notes on Software Engineering, vol. 4(2), p.157, 2016.
21. Rahman, Md, and Shamim Ripon, "Elicitation and Modeling Non-Functional Requirements-A POS Case Study," arXiv preprint arXiv:1403.1936, 2014.
22. Herrmann, A. and Paech, B, "MOQARE: misuse-oriented quality requirements engineering," Requirements Engineering, vol. 13(1), pp.73-86, 2008.
23. Mala, GS Anandha, and G. V. Uma, "Elicitation of non-functional requirement preference for actors of usecase from domain model," InAdvances in Knowledge Acquisition and Management, Springer Berlin Heidelberg pp. 238-243, 2006.
24. Sun, Jie, Liping Zhao, Pericles Loucopoulos, and Bo Zhou, "Qra: A quality requirements analysis approach for service systems," In Services Computing (SCC), IEEE International Conference on, pp. 25-32, 2013.
25. Chung L., Nixon, J. M. B. and Yu, A. Non-functional Requirements in Software Engineering. Springer, Reading, Massachusetts, 2000.
26. Domah, D. and Mitropoulos, F.J., "The NERV methodology: A lightweight process for addressing non-functional requirements in agile software development," In SoutheastCon, IEEE, pp. 1-7, 2015
27. Mead, Nancy R., and Ted Stehney, "Security quality requirements engineering (SQUARE) methodology," ACM, Vol. 30, No. 4, 2005.
28. Burgess, Christopher, Aneesh Krishna, and Li Jiang, "Towards optimising non-functional requirements," In Quality Software, QSIC'09, 9th International Conference on, IEEE, pp. 269-277, 2009.
29. Andreopoulos, Bill, "Achieving Software Quality Using the NFR Framework: Maintainability and Performance," In Proc. 3rd Int'l Conf. Computer Science, Software Eng., Information Technology, e-Business, and Applications, 2004.

30. Li, Feng-Lin, Jennifer Horkoff, John Mylopoulos, Renata SS Guizzardi, Giancarlo Guizzardi, Alexander Borgida, and Lin Liu, "Non-functional requirements as qualities, with a spice of ontology," In Requirements Engineering Conference (RE), IEEE 22nd International, pp. 293-302, 2014.
31. Guizzardi, R., Li, F.L., Borgida, A., Guizzardi, G., Horkoff, J. and Mylopoulos, J., "An ontological interpretation of non-functional requirements," In Formal Ontology in Information Systems: Proceedings of the Eighth International Conference, Vol. 267, p. 344, IOS Press, 2014.
32. Jingbai, T., Keqing, H., Chong, W. and Wei, L., "A context awareness non-functional requirements metamodel based on domain ontology," In Semantic Computing and Systems, WSCS'08, IEEE International Workshop on, pp. 1-7, 2008.
33. Moreira, A., Araújo, J. and Brito, I., "Crosscutting quality attributes for requirements engineering," In Proceedings of the 14th international conference on Software engineering and knowledge engineering, ACM, pp. 167-174, 2002.
34. Gnaho, C., Semmak, F. and Laleau, R., "Modeling the Impact of Non-functional Requirements on Functional Requirements," In Advances in Conceptual Modeling, Springer International Publishing, pp. 59-67, 2014.
35. Song, X., Duan, Z. and Tian, C., "Non-Functional Requirements Elicitation and Incorporation into Class Diagrams," In Intelligent Information Processing, Springer Berlin Heidelberg pp. 72-81, 2010.
36. Casamayor, A., Godoy, D. and Campo, M., "Identification of non-functional requirements in textual specifications: A semi-supervised learning approach," Information and Software Technology, vol. 52(4), pp.436-445, 2010.
37. Rago, A., Abait, E., Marcos, C. and Diaz-Pace, A., "Early aspect identification from use cases using NLP and WSD techniques," In Proceedings of the 15th workshop on Early aspects, ACM, pp. 19-24, 2009.
38. Sampaio, A., Chitchyan, R., Rashid, A. and Rayson, P., "EA-Miner: a tool for automating aspect-oriented requirements identification," In Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering, ACM, pp. 352-355, 2005.
39. Kassab, M., Ormandjieva, O. and Daneva, M., "An Ontology based approach to non-functional requirements conceptualization," In Software Engineering Advances, 2009. ICSEA'09. Fourth International Conference on, IEEE, pp. 299-308, 2009.
40. Dobson, G., Hall, S. and Kotonya, G., "A domain-independent ontology for non-functional requirements," In e-Business Engineering, ICEBE, IEEE International Conference on, pp. 563-566, 2007.
41. Al Balushi, T.H., Sampaio, P.R.F., Dabhi, D. and Loucopoulos, P., "ElicitO: a quality ontology-guided NFR elicitation tool," In Requirements Engineering: Foundation for Software Quality, Springer Berlin Heidelberg, pp. 306-319, 2007.
42. de Almeida Falbo, R., de Menezes, C.S. and da Rocha, A.R.C., "A systematic approach for building ontologies," In Progress in Artificial Intelligence—IBERAMIA, Springer Berlin Heidelberg, pp. 349-360, 1998.
43. Dobson, Glen, Russell Lock, and Ian Sommerville, Quality of service requirements specification using an ontology, 2005.
44. Al Balushi, T.H., Sampaio, P.R.F., Dabhi, D. and Loucopoulos, P., "Performing Requirements Elicitation Activities Supported by Quality Ontologies," In SEKE, pp. 343-348, 2006.
45. Hughes, C. and Hillman, J., "Qos explorer: A tool for exploring qos in composed services," In Web Services, ICWS'06. International Conference on, pp. 797-806, 2006.

46. Lee, S.W., Muthurajan, D., Gandhi, R.A., Yavagal, D. and Ahn, G.J., "Building decision support problem domain ontology from natural language requirements for software assurance," *International Journal of Software Engineering and Knowledge Engineering*, vol. 16(06), pp.851-884, 2006.
47. Li, F.L., Horkoff, J., Mylopoulos, J., Liu, L. and Borgida, A., "Non-Functional Requirements Revisited," In *iStar*, pp. 109-114, 2013.
48. Rashwan, A., Ormandjieva, O. and Witte, R., "Ontology-based classification of non-functional requirements in software specifications: a new corpus and svm-based classifier," In *Computer Software and Applications Conference (COMPSAC), IEEE 37th Annual*, pp. 381-386, 2013.
49. Jingbai, T., Keqing, H., Chong, W. and Wei, L., "A context awareness non-functional requirements metamodel based on domain ontology," In *Semantic Computing and Systems, WSCS'08, IEEE International Workshop on*, pp. 1-7, 2008.
50. Ledru, Yves, et al., "An attempt to combine UML and formal methods to model airport security," *CAiSE Forum*, 2006.
51. Snook, C. and Butler, M., "UML-B: Formal modeling and design aided by UML," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol.15(1), pp.92-122, 2006.
52. WordNet (2010) Princeton University [online] <http://wordnet.princeton.edu/wordnet> (accessed 27 November 2015).
53. De Marneffe, M.C., MacCartney, B. and Manning, C.D., "Generating typed dependency parses from phrase structure parses," In *Proceedings of LREC (Vol. 6, No. 2006)*, pp. 449-454, 2006.
54. Unnati Shah and Devesh Jinwala (in Press), "Resolving ambiguity in natural language specification to generate UML diagrams for requirements specification", In *International Journal of Software Engineering, Technology and Applications*, Inderscience.
55. Borgatti, Stephen P., "How to explain hierarchical clustering," pp.78-80, 1994.
56. Duan, Chuan, and Jane Cleland-Huang, "A clustering technique for early detection of dominant and recessive cross-cutting concerns," In *Proceedings of the Early Aspects at ICSE: Workshops in Aspect-Oriented Requirements Engineering and Architecture Design*, p. 1. IEEE Computer Society, 2007.
57. Li, Z., Rahman, Q.A. and Madhavji, N.H., "An Approach to Requirements Encapsulation with Clustering," In *WER*, pp. 92-96, 2007.
58. Belsis, P., Koutoumanos, A. and Sgouropoulou, C., "PBURC: a patterns-based, unsupervised requirements clustering framework for distributed agile software development," *Requirements Engineering*, vol. 19(2), pp.213-225, 2014.