MACHINE LEARNING (ML) FRAMEWORK FOR IDENTIFYING INCONSISTENCY IN SOFTWARE REQUIREMENT DOCUMENTS (SRD)

Tamaike Brown
Department of Computer Science, North Dakota State University, Fargo, ND 58102, USA
Tamaike.brown@ndus.edu

Abstract

The use of Natural Language (NL) in software requirement specification (SRS) documents introduces inconsistency. Numerous tools and methods are available for managing requirements. However, there are few procedures and tools that provide support for analyzing inconsistency in SRS textual documents. Detecting inconsistencies in SRS document is a challenging task that has spark the interest of researchers. The primary methods used to identify inconsistencies in NL SRS document are reviews and inspections. However, the application of human labor is time consuming, ineffective and introduces difficulties.

This paper describes a framework for identifying inconsistency in SRS documents. The proposed tool incorporates a knowledge base system that integrates the semantic and syntactic analysis of requirements.

CCS Concepts:
• Software and its Engineering → Software Functional Properties • Correctness → Consistency

Keywords
Inspection, inconsistency

1. Introduction

The first stage of the software development life cycle (SDLC) is the requirement phase. This stage involves eliciting requirements from stakeholders in natural language (NL), analyzing requirements, negotiating and revising requirements before all stakeholders sign off on the final document. That is, the software requirement specification document, SRS [31]

Requirement elicitation, the process of gathering and combining systems functionalities from stakeholders is the first phase of the requirement stage and has a direct impact on the quality of the software produced [13] [14] [16]. It is imperative at this stage that intensive human communication occurs in order to gather the correct requirements [15]. Subsequently a software requirement specification document is developed. This document captures all the required aspects of the system including functional requirements. At one end of the pendulum, stakeholders should be able to specify their needs as it relates to the system under development. On the other end, there must be some automated inspection procedure in place to check the SRS document for inconsistencies. Because SRS is the fundamental building block that guides the development process and this document should be free of defects such as ambiguity, incorrectness and must be analyzed as it pertains to consistency, completeness and correctness before all stakeholders sign off in agreement with its content [17]. To add, according to authors Xowghi and Gervasi addressing consistencies in SRS document directly has an impact on reducing incompleteness and incorrectness [18]. Nelson and his co-authors discovered that software developers design systems that contain embedded errors because these systems are dependent on the system requirements [32]. Errors in requirements are expensive to fix, increase development time due to rework and result in maintenance issues. According to Nelson:

"*A software error costing a mere $1 when caught early in the life cycle, cost $5 to correct at midpoint and $100 to correct later in the life cycle"*

Based on Nelsons statement it is imperative that early detection of inconsistency is done in order to reduce overall SDLC related cost as well as to improve product quality.

As the utilization of software continues to proliferate in today's society, software quality is becoming increasingly a paramount issue. As a matter of fact, with the exponential increase in software applications, inspection has equally grown as a vital activity for the purpose of validating requirement specification documents. Software inspection has been in existence for over three decades to ensure specifications and domain properties are equivalent to the requirements thus, Broy and Denert (2002) rightly stated:

"Inspections are now 30 years old and they continue to improve software quality and maintainability, reduce time to delivery and lower development cost"
[M. Broy & E. Denert, 2002 pp. 215] [11]

Software engineering literature consistently designates inspection and testing as two widely renowned activities for software quality improvement. However, while both methods are used for defect detection (such as inconsistencies, ambiguity, and incompleteness) and removal in software products, testing can only be done when the software has been implemented. On the other hand, inspection (manual or automated) is the only method that can be applied in the early stage of the software development to prevent rework of application requirements, design and code by finding defects and removing them [7] [8] [9] [10].

Even though inspections have been utilized for more than three decades, it is quiet alarming that software is still being released with defects [1]. For that reason, further innovative research is needed to discover more real-world, simple, effective and automatable inspection methods [12].

James Martin (1984) stated, "The root cause of 56 percent of all defects identified in software projects are introduced during the software requirement stage of the development life cycle" [6]. This may be because requirements are gathered through NL [3]. The possibility of dissimilarities in interpretation and understanding by various stakeholders can expeditiously lead to defects such as ambiguity, inconsistency, incorrectness, omission and superfluous information, just to name a few [1]. Hence, according to Alshazly *et al.*, it is vital to detect these defects in the interest of conserving resources and the conformance to requirements. In addition, it is necessary to uncover defects to preserve software quality and reliability by removing bugs [1]. Kamalrudin also noted that one of the major contribution to imprecision or defects in software requirement specification document is the use of natural language [2]. According to Fabbrini (2001) *et al*, SRS documents tend to contain errors because of the use of NL when translating requirements from stakeholders [4].

Numerous documented research focus on identifying inconsistencies in software requirements whether the model use is formal (example Zed language / Z notation) or semi-formal (example NL). For instance, Grant et al, used Z notation specification to validate and verify functional requirements for safety critical systems [5]. However, most of the studies ignore or does not define an automated procedure for inspecting textual document for inconsistencies.

Therefore, our research set out to define a procedure to identify inconsistencies in software requirement specification that can be automated for future studies, with particular emphasis on the functional requirements.
Although there are many definitions for Inconsistency, in the context of this work, we will utilize the description of Inconsistent as defined by Alshazly *et al.*, to be:

Any part of the functional requirements document that is inconstant with other related functional requirements of the same type, structure or with the problem that the SRS artifacts solves [1]. Inconstant with regards to use of words, terminology, and internal logics [25] [26] [27].

This study is motivated by results of prior research whose limitations spark the need to develop a new methodology to detect inconsistency. Our work takes into account some properties of the studies reviewed as well as built on their limitation to create an improved approach.

2. Related Works

In this segment, we presents previous studies on identifying inconsistencies in SRS document.

Paper	Approach	Result
Kamalrudin [2]	Develop an automated tool for detecting inconsistencies between textual SRS and use case (use case are generated from interactions list)	The evaluation of the study concluded that the researchers were able to trace natural language requirements to a set of abstract interactions. Interactions (use case) are a compilation of phases extracted from the natural language. Nonetheless, the tool was not able to exhaust all possible forms of inconsistencies, since traceability is difficult [20]. The tool requires more work be done in order to fully address the need to reduce inconsistences in requirements. For example, the need to have a library to extend the opportunity of locating consistent interactions.
XLinkit [19]	First order logics, object Z specifications, specification tests, model abstraction and model checking to verify requirements. In order to detect inconsistencies goal elaboration, order abduction and morphing of path (knowledge & rule base)	The tool was able to manage consistency between software artefacts generated at each stage of the software development life cycle.
Sugimoto *et al.*[21]	Requirement Framework Model to detect inconsistency of SRS. Application of the Dempster and Shafer's theory was applied to interpret the inconsistency.	Researchers were able to locate 2 inconsistent requirement sentences of 46 sentences
Koth *et al* [28]	Heuristics Approach with specification requirement and semantics	The use of semantic checker as an incremental evaluation approach efficiency was improved. That is, the methodology prevents the re-evaluation of the entire XML document when new semantics information is added about attributes, rather it uses the previous saved semantics computation to make comparison to the new modification.

Table 1: Summary of Approaches to Detecting Inconsistencies in Software Requirement Specification Document

In an effort to document the gap that exist in detecting inconsistencies in software requirements Kamalrundin noted a deficiency in the inconsistencies tools literatures, in terms of the need to assess different representation and check for inconsistencies between scenario and textual descriptions [2]. The researcher proposed a consistency management and tracing tool within the Eclipse-based Marama meta-development environment that is able to trace requirements back to their design representation, particularly the use case diagram [2]. In the authors' analysis, the innovative tool should assist in correcting requirements during the translation process by identifying inconsistencies between the NL requirements and interactions. The proposed methodology incorporated four steps. (1) Collection of NL requirements (2) Analysis is performed on NL requirements using a database comprising of use case interactions (3) use case models are generated and (4) items are selected from the use case and compare to the NL [2]. While the author focused on scrutinizing inconsistencies of requirements, yet again another study only looks on keeping requirements consistent between informal (NL) and semi-formal (use case) requirements. Moreover, no mention was made on the protocols that govern identifying appropriate generated use case models to compare to NL.

Kozlenkov *et al* developed a knowledge based tool with the capability of ensuring consistency between different artefacts produced at different stages in the software development life cycle with main focus on large and complex systems. In addition, the instrument ensures that the system under development meets the functional requirements [19]. One of the significant shortcoming of the study is the use of formal specifications. In order for users to make use of formal specifications they need to have a detail knowledge of the modeling language or have the language explain to them continuously. In addition, while the researchers were busy checking for inconsistencies between models, there has been no method incorporated in their tool to first check for inconsistencies within the SRS document. Since all models are based on the SRS, which is the driving force of the system, it is very necessary and imperative that this document is thoroughly check for such aforementioned defects. So as to prevent any carrying over of defects (inconsistency) due to the use of NL.

Other researchers utilize techniques such as domain ontology and ontology Library Management System to detect and correct inconsistencies between NL and other analysis and design representation [23] [24].
In 1999 Sugimoto proposed a static technique to locate and interpret inconsistency of SRS documents [21]. The study defined inconsistencies in two forms; 1. Illegal use of words and 2. When two or more requirement sentences that define the same data structure differs. The authors implement their methodology my making use of the requirement frame model to detect inconsistencies of SRS [21]. The operation of the requirement frame model integrates the following concepts:

- Identifying objects (verbs) and object types (attributes)
- Define operations among objects and role of the operations (cases) where cases represent concepts about agents, objects, goals of operations.

Based on the concepts, three major frames were formed [21] [22].

- A noun level frame

- A sentence level frame
- A functional frame

In order to determine the feasibility of the technique the authors developed a text-base requirement language name X-JRDL that is based on the requirement frame model. The purpose of X-JRDL dictionary is to analyze and simplify sentences that is transferrable into CRD (conceptual requirement description). X-JRDL description (words such as pronouns, verbs and adjectives) is analyzed using three interpreters. Verbs are further categorized into different concepts to find illegal use of words which the researchers categorized as a form of inconsistencies. Subsequently the defined procedure was applied to a stock control system. Of the 46 sentences in the SRS two (2) inconsistencies were identified [21]. While giving credit to the researchers for developing such novel approach the study falls short in the following aspects:

- It is difficult to prove the successfulness of the approach if readers are unable to identify the actual number of inconsistencies in the document use to support the study.
- The dictionary was written in Japanese language, hence that limits the ability to accept the study in a general scenario.
- The dictionary excludes the use of noun instead pronouns are incorporated in the X-JRDL. This is undesirable since a SRS document and any translation thereof must be specific about the actors who should perform particular requirements as define in the SRS document. Not being specific about who or what should perform system functionality may lead to yet another form of inconsistency in the SRS.
- The methodology requires automation in order to further identify any anomaly in the proposed procedure and enhance the technique.
-

On the other hand, the results additionally provided insights into how the proposed method can be ameliorated for automation.

Koth *et al.*, [28] developed a methodology based heuristic analysis approach coupled with requirement specifications and semantics. This approach was selected by the researchers because in comparison with its counterpart formal analysis, heuristic analysis does not require the structure of mathematical model for making decisions [29] and is capable of specifying the steps for achieving required goal [30]. In addition, this is the most commonly used technique for identifying consistencies or inconsistencies in requirements. The semantic incremental attribute evaluation technique was developed and used on Extensible Markup Language (XML) document. This approach introduces incremental facilities and evaluates the attributes associated to XML semantics by adding incremental strategy to XML semantic evaluator [28]. In order to check the consistency of documents repeatedly until a consistent document is produced the Propagate Algorithm was also employed [28].

Gnesi *et al.* [33] developed and implemented a tool so called Quality Analyzer for Requirement Specification (QuARS) for analyzing lexical and syntax in software requirement documents. The function of the tool is to extract structured information and metrics for detecting linguistic inaccuracies and defects that can result in ambiguity at the later phases of the software development process. The lexical component looks on a word

in a language (example strong) while the syntactic component concentrates on the arrangement of words in a sentence (example, the users should enter website with password. – This can be interpreted as the website is only accessible when a user enters a password or that a website that has a password is only accessible to the user. Because QuARS is limited to defect identification and readability an updated version of QuARS was released named QuARS Express by Bucchiarone *et al* that incorporate improvement on defect identification and readability analysis [34].

3. Proposed Approach

We are proposing a methodical approach to our new hybrid technique. Figure 1 shows our proposed approach. The process of identifying inconsistencies in the SRS document will begin with comparing the requirements of each bucket to the consistency attributes or rules stored in the knowledge base database from a semantic and syntactic point of view. The semantic looks on the meaning of words in a language. While the syntactic concentrates on the analysis of requirement statement constructs. Once a rule has been broken, the tag for that requirement will be recorded. Subsequently, a log file will summarize each rule that has been broken that is associated with a particular requirement.

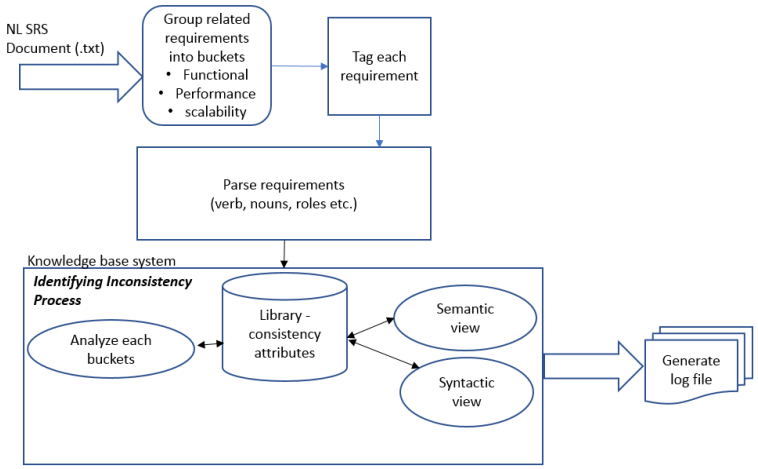

Figure 1: Proposed Approach

Our proposed technique involves the following steps:

- Loading a textual SRS document to the system.
- Grouping related requirements into different buckets. The system should be capable of recognizing requirement ID such as 1.1.a. The goal of grouping relate items into different buckets is to facilitate effective and efficient analysis of the data as well as to reduce re-evaluation processing time. In addition, this will help to record analysis result such that all inconsistency rule pertaining to a requirement appears together.
- Tag each requirement in a bucket: - A suffix associated with a bucket will be added to the tag of each requirements. The goal is to refine the scope of re-evaluation of the SRS document after modification. For example, if a change is

made to a particular section of the SRS document, only that specified bucket will be evaluated after the requirement had been added to the bucket rather than all buckets. Also, tagging each requirement provides identification of consistency rule violation traceability from the machine learning knowledge based system to the initial requirement

- Parse Requirements: - Requirement will be parsed from both a syntactic and semantic standpoint. As outlined earlier on the syntax component will focus on the construct of a sentence. Syntactic parsing will highlight the part of speech such as verb or noun of a word in a sentence as well as the role of each word. Meanwhile the semantic parsing of the document will identify specific words and phrases in sentences of the SRS document.
- In our approach we put forward the need for inclusion of a machine learning knowledge base system (MLKBS). So that the system will be able to draw upon the knowledge of human expert to identify inconsistency issues that normally require human competence. The knowledge base system (MLKBS) will be directly identifying and locating inconsistencies by making use of machine learning systems with Meta data for storing all the information related to requirements and rules.
    - The MLKBS incorporate a comprehensive Meta dictionary which is a collection of facts about the system's domain within the database to check for verbs, nouns etc.
    - The MLKBS should be able to provide reasoning about information in the knowledge base by identify sentences and performing comparison amongst sentences syntax and semantics rules.
    - The system must be able to store values for check attributes to reduce validation time when modification is done to the SRS document.

The machine learning system will make use of all the stored data in Meta data for further data mining and automated reasoning.

```
// Identifying Inconsistencies in SRS Document Algorithm
load_doc();

for each requirement
    {
    group_related_req();
    add_to_bucket();
    }

add_tag();
parse_req();

while (bucket >=1)
    {

    select_bucket();
    analyze_sem_view();
        con_attributes();
    analyze_syn_view();
        con_attributes();

    }
gen_log_file();
```

Figure 2: Algorithm for Identifying Inconsistency in SRS document

Number of inconsistency identify (*NoI*) $= \sum_{i=1}^{n} b$
Where *b* represents buckets
Where *n* represents number of buckets

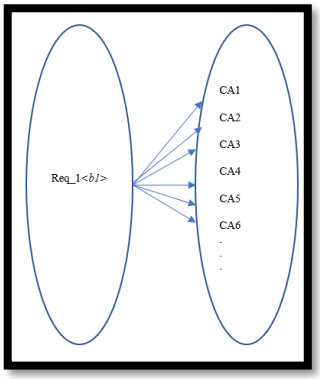

Figure 3: Mapping a Requirement to Consistency Rules in the Knowledge Base System

Figure 3 illustrates how each requirement will be compared to the consistency rules stored in the library of the KBs.

4. Case Study

The case study below shows from an abstract level how the automated system should function. We assume the development of a website that allows users to make purchasing of items online. Our customer requires that only users with authorized access – user name and password are allowed to use the website. We are assuming that our customer means well when stating the needs of the business. Initially, we do not have a clear knowledge about the domain, table 2 and table 3 shows our beginning requirements and consistency attributes.

Requirements ID	**Requirement Description**	**Requirement tag Number**
Req 1.0	The user shall enter website with a user name and password	Req_1<*b1*>
Req 2.0	The software shall support database entry	Req_2<*b1*>
Req 3.0	Accessing the website allows purchasing	Req_3<*b1*>
Req 4.0	Users who do not have a user name and password are denied access to the website	Req_4<*b1*>
Req 5.0	Users whose user name and password appears in the database shall be authenticated	Req_5<*b1*>
Req 6.0	Password must be strong containing letters (upper and lower case), numbers and special characters	Req_6<*b1*>

Table 2: Beginning Online System Requirement

Where *b1 represents bucket one* which is the suffix for each tag

Requirement Tag Number	**Consistency Attribute ID**	**Consistency Attributes (syntactic and semantic Rules)**
Req_1<b1>	CA1	If user enter user name and password then allow user access to the website
Req_2<b1>	CA2	If database exist then user data shall be stores in the database
Req_3<b1>	CA3	If user is authenticated then user will be allowed to purchase
Req_4<b1>	CA4	If user does not have a user name or password then user will be prevented from entering the website
Req_5<b1>	CA5	If user name and password exists in the database then the user is valid
Req_6<b1>	CA6	If password contains upper and lower case letters, numbers and special characters then it is strong

Table 3: Consistency Attributes

For demonstration purpose, we are only using one bucket (***b_1).*** Each requirement will be compared to all the constraint attributes that exist in the knowledge base system.

Requirement tag number	**Consistency Attributes**					
	CA1	CA2	CA3	CA4	CA5	CA6
Req_1<b1>	Y	N	N	N	N	N
Req_2<b1>	N	Y	N	N	N	N
Req_3<b1>	N	N	Y	N	N	N
Req_4<b1>	N	N	N	Y	N	N
Req_5<b1>	Y	N	N	N	Y	N
Req_6<b1>	N	N	N	N	N	Y

Table 4: Identifying Inconsistency Result Table

Log File	
Req_1<*b1*>	is inconsistency with CA2, CA3, CA4, CA5, CA6
Req_2<*b1*>	is inconsistency with CA1, CA3, CA4, CA5, CA6
Req_3<*b1*>	is inconsistency with CA1, CA2, CA4, CA5, CA6
Req_4<*b1*>	is inconsistency with CA1, CA2, CA3, CA5, CA6
Req_5<*b1*>	is inconsistency with CA2, CA3, CA4, CA5
Req_6<*b1*>	is inconsistency with CA1, CA2, CA3, CA4, CA5
NoI in ***b1*** = 29	

Table 5: Resulting Log File

5. Conclusion And Future Work

We defined a new approach for identifying inconsistencies in SRS document. This approach has taken into consideration strengths of previous study such as Gnesi et al, Sugimoto and Koth et al. It builds on those systems techniques shortcomings and it incorporates the use of knowledge based system.
Future work is required to build the system or find means to integrate systems together that will identify inconsistency in SRS document as outlined by this approach.

6. References

[1] Alshazly, A.A.; Elfatatry, A.M.; Abougabal, M. S. Detecting Defects in Software Requirements
Specification. Alexandria Engineering Journal,
2014 53, Pages 513–527 http://dx.doi.org/10.1016/j.aej.2014.06.001

[2] Kamalrudin, M. Automated Software tool for checking the inconsistencies of Requirements. IEEE/ACM International Conference on Automated Software Engineering (2009)

[3] Aceituna, D.; Hyunsook Do; Walia, G.S.; Lee, Seok-Won. Evaluating the use of model-based requirements verification method: A feasibility study. Empirical Requirements Engineering (EmpiRE), 2011 First International Workshop on Year: 2011 Pages: 13 - 20, DOI: 10.1109/EmpiRE.2011.6046248 IEEE Conference Publications

[4] Fabbrini, F., et al. The linguistic approach to the natural language requirements quality: benefit of the use of an automatic tool. in Software Engineering Workshop, 2001. Proceedings. 26th Annual NASA Goddard. 2001

[5] Grant, E.S; Jackson, V.K.; Clachar, S.A.; Towards a Formal Approach to Validating and Verifying Functional Design for Complex Safety Critical Systems. Journal on Computing (JoC) Vol.2 No.1, April 2012.

[6] Martin, J. An Information Systems Manifesto.
Prentice Hall, 1984

[7] Fagan, M. Design and Code Inspections to reduce Errors in Program Development, IBM System Journal 15 (3) 1976) 182 – 211.

[8] Fagan, M., Advances in Software Inspection, IEEE Transactions on Software Engineering 12(7) (1986) 744-751.

[9] Ackerman, A.F.; Buchwald, L.S.; Lewski, F.H. Software Inspection: An effective verification Process, IEEE Software 6(3) (1989) 31-36

[10] Hetzel, B., The Complete Guide to Software Testing, John Wiley & Sons, USA, 1988.

[11] Broy, M.; Denert E. Software Pioneers: Contribution to Software Engineering. 2002

[12] Parnas, D.L.; Lawford, M. The Role of Inspection in Software Quality Assurance, IEEE Transaction on Software Engineering 29(8) (2003) 674-676.

[13] Somerville, I. Requirements Engineering: A Good Practice Guide. New Delhi, India: Wiley 2009

[14] Aurum, A.; Wohlin, C., Engineering and Managing Software Requirements. New York, NY, USA: Springer, 2006.

[15] Anwar, F.; Razali, R.; Ahmad, K. Achieving effective communication during requirement elicitation – A conceptual framework. In Proceeding 2nd Int. Conf, Softw. Eng. Comput. Syst., 2011, pp. 600-610

[16] Gerald Kotonya, I.S. Requirement Engineering Process and Techiques, ed P.P.W. Professor Davis Barron. 1998, West Sussex England: John Wiley & Sons Ltd 282

[17] Denger, C.; Berry D.M.; Kamsties, E. Higher Quality Requirements Specifications through Natural Language Patterns, in Proceedings of the IEEE International Conference on Software-Science, Technology \& Engineering. 2003, IEEE Computer Society. p. 80

[18] Zowghi, D.; Gervasi, V. *On the interplay between consistency completeness, and correctness in requirements evolution.* Information and Software Technology, 2003. 45(14): p. 993-1009

[19] Kozlenkov, A.; Zisman, A., *Are their Design Specifications. Consistent with our Requirements?* In Proceedings of the 10th Anniversary IEEE Joint International Conference on Requirements Engineering. 2002, IEEE Computer Society. p. 145-156

[20] Aalinoja Julo, OM. Software requirements implementation and management. Software && systems engineering and their applications 2004 vol 1(3). Pp. 1.1 – 1.8

[21] Sugimoto, H.; Ohnishi, A. A detecting and Interpreting Method of Inconsistency of Software Requirement Specifications. 1999, IEEE

[22] Ohnishi, A. Software Requirements Specification Database based on Requirements Frame Model. Proc. IEEE International Conference on Requirements Engineering (ICRE '96), 1996, pp. 221-228

[23] Patel, K.; Gandhi, S., Inconsistency measurement and remove from Software Requirement Specification. 2014, IJEDR (International Journal of Engineering Development and Research). Vol 2(2)

[24] Zhu, X.; Jin, Z. Inconsistency measurement of Software Requirements Specification: An Ontology-Based Approach.Proc. IEEE International conference on Engineering of complex computer systems, ICECCS 2005, pp 402-410

[25] Kamalrudin, M.; Sidek, S., Software Requirements Validation and Consistency Management. International Journal of Software Engineering and Its Applications. Vol. 9, No 10 (2015), pp. 39-58

[26] Zowghi, D.; Gervasi, V., "On the interplay between consistency, completeness and correctness in requirements evolution", Information and Software Technology, vol. 45, (2003), pp. 993-1009

[27] Satyajit, A.; Mohanty, H.; George, C., "Domain consistency in requirements specification," Proc. Fifth International Conference on Quality Software 2005 (QSIC 2005), pp. 231-238

[28] Koth, Y.; Gondow, K.; Katayama, T., "An incremental evaluation approach to check the consistency of XML documents", Proc. IEEE International Conference on Systems, Man and Cybernetics 2002, vol. 6.

[29] Silver, E.A., "An overview of Heuristic Solution Methods", The Journal of the Operational Research Society, vol. 55, (2004), pp. 936-956.

[30] Sutcliffe, A.A.G.; Maiden, N.A.M., "Bridging the requirements gap: policies, goals and domains", Proc. Seventh International Workshop on Software Specification and Design, (1993), pp. 52-55

[31] Malhotra, R.; Chug, A.; Hayrapetian, A.; Raje, R. "Analyzing and evaluating security features in software requirements. International Conference on Innovation and Challenges in Cyber Security (ICICCS-INBUSH), 2016.

[32] Nelson, M; Clarke, J; Spurlock, M.: Curing the Software Requirments and Cost Estimating Blues. PM-Nov-Dec, 1999

[33] Gnesi, S., Fabbrini, F., Fusani, M., and Trentanni, G.: An automatic tool for the analysis of natural language requirements, CRL Publishing: Leicester, 2005.

[34] Bucchiarone, A., Fantechi, A., Gnesis, S., Lami, G., and Trentanni, G.: QuARS Express – An Automatic analyzer of natural language requirements, Proceeding of 23rd IEEE/ACM International Conference on Automated Software Engineering, pp. 473-474, 2008.