# TOWARDS ASSESSING THE QUALITY OF FUNCTIONAL REQUIREMENTS USING ENGLISH/SPANISH CONTROLLED LANGUAGES AND CONTEXT FREE GRAMMAR

Carlos Huertas
Department of Computer Science
Autonomous University of Baja California
Tijuana, México
chuertas@uabc.edu.mx

Reyes Juárez-Ramírez
Department of Computer Science
Autonomous University of Baja California
Tijuana, México
reyesjua@uabc.edu.mx

***Abstract*—Most software applications are derived from customer requirements, which mainly are expressed as functional requirements. These requirements are the basis for software construction; therefore they have an important impact on overall software quality. As software grows so does the requirements, the increase in information often leads to overlooked requirement evaluation due to the amount of effort and time required. One important cause of this complexity in requirements evaluation is the fact that until now, most of them are still written in natural language which inherent all its problems to the requirements. In this paper we extended our previously proposed software requirement model which is based on context free grammar to support both English and Spanish under a controlled context for automated requirement evaluation using natural language processing, besides we provide comparative testing for both languages as well as comparison results for manual and automatic approaches.**

***Keywords-Natural language processing; software engineering; software quality; software requirement***

## I. INTRODUCTION

Regardless of the methodologies used in a software development, there is a step that cannot be missing which is software requirements. Some agile methodologies have minimal planning and analysis but still have to deal with requirements of some kind [1]. According to Boehm and Turner [2] each development methodology has its own *home ground*, and the required methodology to achieve extreme quality is only attained by formal methods.

In the area of software engineering there is still a lack of formalization and researchers have found that requirements are an important cause of project failure mainly because they are expressed in natural language [3-4], which is an informal style. The use of formal languages like UML has been tried to avoid the inherent problems of natural language such as *incompleteness* and *ambiguity*, due to the fact that these problems can cause millionaire defects [5]. However, recent investigations have shown that natural language is still the most used way to write software requirements due the fact there is broader number of people that can understand natural language over formal languages [6]. As a reference, according to the Ethnologue publication [28], English and Spanish are the most spoken languages only after Mandarin, and both alone have a native number of speakers of over 741 million, and represent the third and second most spoken languages in the world. According to the Association of Spanish Language Academies, the Spanish language is the official language of 22 countries, and the most spoken language in the whole America continent, this scenario creates a huge amount of project stakeholders with Spanish as their native language, even in countries like United State with a different official language, official public documents are provided in Spanish too [34].

However, a software requirement has more problems than the inherited from natural language, for a requirement to be considered correct, we can use the quality attributes proposed by Davis [7] as well as the IEEE 830-1998 standard [8] which states that a requirement must be: *unitary, complete, consistent, atomic, traceable, current, feasible, unambiguous, important* and *verifiable*. Our research focuses on three of these quality attributes, which are:

- Complete
- Atomic
- Unambiguous

These have been proved as the attributes that most often cause problems [5] [9-12]. The unintended ambiguity has caused so many problems that it is often referred as, *the Achilles heel* of software requirements specifications [11].

According to Nikora [6] the main problem in software requirement validation is not the fact to review the requirement itself, but the amount of resources required to review a big amount of requirements when the software grows to a considerable scale, given this and the current demand for having systems running quickly [13], there is a need for formal and automated approaches to reduce the required time to check requirements and eliminate the human error prone factor from the requirements phase. All these efforts need to be done in the early stage of development because finding problems later in the cycle costs much more to fix [14].

In this paper we review our model for functional requirements based in a Controlled Natural Language (CNL). We have reduced the grammar, but the language dictionary remains the same, in this work we have extended the model to work with Spanish language as usually only English is supported[29], including our previous works [24][25]

The proposed solution overview is a formalization for software functional requirements, a CFG in Backus Naur

Form (BNF) notation built upon regular expressions helps to deal with the English and Spanish CNL. A set of identifiers for non-atomic and pragmatic ambiguous requirements were developed and with the aid of a previously developed tool named NLARE and Natural Language Processing (NLP) techniques it is possible to propose an automatic, fast and consistent environment for requirement evaluation.

The performed tests shows the progress with the Spanish model when compared with the English model as well as accuracy and speed comparison with manual vs. automatic approach.

The remaining of this paper is structured as follows: In section 2, we provide an overview of related works. In section 3, representation of a functional requirement is proposed. In section 4, our proposed solution is detailed. In section 5, we provide a set of test and results. In section 6, threats to the validity of this work are discussed. In section 7, we show our conclusions and future work.

## II. Related Works

The natural language has been used as the best way to communicate between stakeholders, therefore a considerable amount of works have tried to search, fix and minimize inherent errors before they reach the requirements [15]. According to Tjong [16], these approaches can be classified intro three main categories:

- Linguistic rules and analytical keywords
- Guideline rules
- Specific language patterns for a domain.

For the first approach, we can cite the work from Wilson [17] that defines the overall quality attributes in two blocks:

- **Quality indicators:** these are defined as the attributes that provides quality to requirements, such as atomicity, unambiguous, completeness, traceability, consistency among other attributes
- **Lack of quality attributes:** these are all the attributes that represents potential problems such as imperatives, options, weak phrases among others.

The Software Assurance Technology Centre (SATC) created a tool named Automatic Requirements Measurement (ARM) which is based on this quality attributes and indicators, these sets of rules are based on the English language and works on the words themselves, e.g. it searches for phrases like *adequate*, *as appropriate*, *be able to*, etc. because these are considered weak phrases that may cause problems. A similar approach can be distinguished from the work of Fabbrini et al. [18] which states two different aspects of requirement quality, such as requirement sentence quality (RSQ) and requirements document quality (RDQ). Among the RSQ we can find indicators for *implicit subject sentences*, *subjective sentences*, *underspecified sentences*, etc. In RDQ we find *comment frequency*, *readability index*, *unexplained sentences*, etc. These quality indicators were included in a tool called Quality Analyzer of Requirement Specifications (QuARS)[29], these indicators are very similar to the approach in ARM, e.g. it looks for words like *this*, *these*, *that* and *those*, because these can cause misunderstandings, besides they check for the appearance of multiple main verbs or subjects as these are considered problem indicators too. In summary we can see that ARM and QuARS works in a similar way as both work with indicators at word level. In our work we differ by the fact that we are working with the requirement grammar elements, a more detailed review of this will be explained in section 4.

For the guideline rules approach we can review the work from Gotz and Rupp [19], they propose three main transformation process to model the original intention of a person to communicate the requirement meaning; these transformations are:

- Deletion
- Generalization
- Distortion.

Authors provide a description for each transformation which is as follows: *deletion* reduces the perception of a person, with missing information it is easier to get lost from the real intended meaning; *generalization* assumes that the reader has a given experience to understand a context; and *distortion* uses nouns for complex process e.g. the take off. According to the authors these transformations are the base for defects in requirements. Among other approaches of this kind, there is the classification of requirements in static and dynamic which is proposed by Juristo et al. [20], they have defined structured languages called Static Utility Language (SUL) and Dynamic utility Language (DUL); this approach helps to reduce ambiguities by restricting the level of freedom in requirement writing, an important difference with our work is that we do not enforce a set of words or grammar, but instead provide guidelines about how to write the requirement in terms of expected elements but without vocabulary boundaries, besides we accept requirements in Spanish too as we do our processing at grammar level.

In the language pattern for domain approach there is the work from Ohnishi [21] who presents a text based language called eXtended Japanese Requirements Description Language (X-JRDL) which is based on a frame model of *nouns*, *cases* and *functions* but restricted to the file system domain only, according to author when these requirements are reviewed by the analyzer the system can review each requirement description. In our work we focus on functional requirements but do not restrict a domain, and we provide support for English and Spanish instead of Japanese. A similar approach is the one from Rolland and Proix [22] who developed a tool called OICSI, which uses linguistic patterns to work especially in the database domain. These patterns are divided in elementary and sentence, but unlike Ohnishi work, these are based on French language.

In summary, we can conclude four key differences from the previous works.

- All our evaluations are based on grammar-level
- There are no restrictions for words or domains
- Rules are based on regular expressions
- We propose guidelines for requirements
- Multi Language support English and Spanish

## III. FUNCTIONAL REQUIREMENT FORMALIZATION

In this work, we focus only on functional requirements, specifically the ones written in natural language. Our formalization process begins with the fact that a requirement is a statement and in linguistics this is a sentence [23]. A sentence then is a set of words; therefore we conclude that a requirement is finite set of words denoted by $W_1$ to $W_n$ as shown below.

*R = "The system will check if password matches user in DB"*
$W_1$ $W_2$ $W_3$ $W_4$ $W_5$ $W_6$ $W_7$ $W_8$ $W_9$ $W_{10}$

Given this, we can then propose or first relation about requirements and words.

$$R= \{W_1,\ldots, W_n\} \quad (1)$$

However, as part of the process and based on the previous example we can infer that $W_1$ and $W_2$ are not the same kind of words, and therefore a formal representation of these differences needs to be addressed. Under this research we have defined a word W as a tuple of elements *t* and *M*, which are explained below.

$$W_i = (t, M) \quad (2)$$

In previous relation (2) we define the elements as:

- *t*: identifier for word lexical group.
- *M*: set of potential interpretations for a word.

With this relation, we can now state a difference between words based on the lexical group they belong, e.g. *nouns*, *verbs*, *determiners*, etc. and the potential meanings or role that they could have played in a given sentence. It is important to notice that some words could belong to a different lexical group. Let us take for example the word "*race*" which can be a noun or a verb and to set the right group, a context revision needs to be done. Now if we replace the definition of (2) in (1) we get the final formal representation of how a functional requirement is constructed in terms of its elements as shown below.

$$R = \{(t, M)_1, \ldots (t, M)_n \} \quad (3)$$

Up to this point, we have defined the overview of the requirement in terms of elements, but to recognize these set of words as a functional requirement we require that among its content at least three elements can be identified.

- Actor: performs the action in the statement.
- Function: what action needs to be performed
- Detail: action expected under what conditions

The conclusion to expect at least these elements is based on previous works [24-25] and are basically based on heuristics determined by an evaluation of large amount of university projects and the information usually required by the developers or quality assurance engineers in further stages of project life cycle. Based on this requirement formalization we propose a solution to reduce requirement evaluation time and errors, which will be explained in the next section.

## IV. PROPOSED SOLUTION

To reduce the resources needed for requirement evaluation, we propose an automated approach that not only improves speed but also eliminates the human error prone factor, this is a very important goal, because as we can learn from Nikora[6], humans, hence manual evaluation it is prone to fail, regardless of efforts, there is always uncertainty about the result if evaluation is performed without aids as natural language processing (NLP) techniques. To achieve this, the upcoming sets of NLP techniques were used. As mentioned before, our research focuses in three main quality attributes, *atomic*, *unambiguous* and *complete*. A representation of the problematic affecting these quality attributes can be then denoted as follows:

Conjunction (lack of atomicity) exist if (4) is true.

$$[\sum W_i(t) = \text{conjunction }] > 0 \quad (4)$$

Ambiguity exists if (5) is true.

$$W_i (M) > 1 \quad (5)$$

Under this research, the element *t* can be a *verb*, *noun*, *pronoun*, *conjunction*, *adverb*, or any other element from the lexical group, when the context defines that the element *t* is a *conjunction* between two sets of words, it causes a lack of atomicity in the requirement and the proper warning is set.

The ambiguity is detected in a similar way, when a word is detected to have more than one potential meaning in the sentence, i.e. the elements of the M set are more than one the warning is set.

Incompleteness exists if any of the sets *actor A*, *function F*, or *detail D* are not found. We discuss these formal representations for the problems in quality attributes more in depth in previous works [24].

In summary we expect each requirement to do not have conjunctions, do not have words that may cause multiple interpretations for a requirement, and have the three main elements mentioned before. As result we can expect a requirement as shown in (6).

R = {{actor A},{function F},{ Detail D}}  (6)

To determine if these elements are present in the requirement, a set of regular expressions were defined that complies with the following pattern to produce a controlled grammar for functional requirements. An example of these expressions is shown below:

- *Actor:*
  *{<DT><NN>+}*
- *Function:*
  *{<MD>?<VB><IN>?<DT>?(<NN>|<NNP>|<NNS>)+}*
- *Detail:*
  *{<IN><DT>?(<NN>|<NNP>)+}*

Using previous definitions as a basis we propose the requirement evaluation as summarized in following figure:

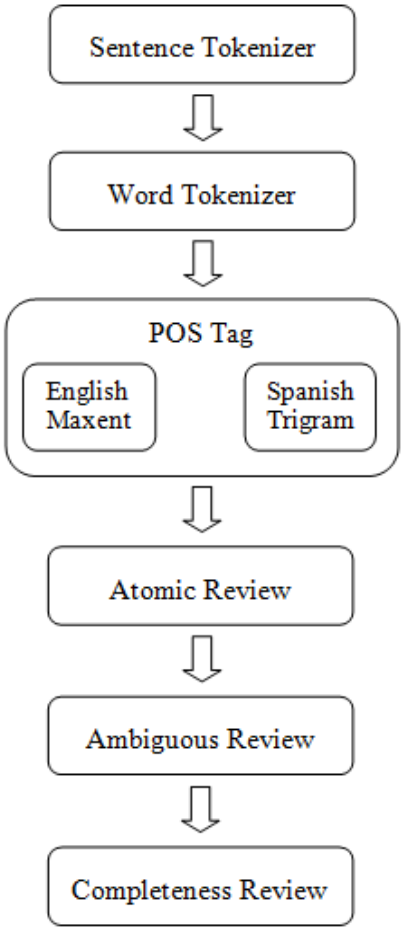


Figure 1. Evaluation overview.

### A. *Sentence Tokenizer*

To check if a requirement is atomic, it is expected to have only one sentence, to evaluate this, the *Punkt Sentencer Tokenizer* is utilized, if a requirement contains more than one sentence it is marked as atomic otherwise a warning will be set.

### B. *Word Tokenizer*

To split the statement in a set of words as in (1), the *Treebank Word Tokenizer* is used to perform this task.

### C. *POS Tag*

To review context and find out the corresponding lexical group for a given words in the requirement, a Part-of-Speech (POS) tagging process is performed, and this specific step increase the complexity as one of our goals is to support both English and Spanish, the Natural Language Tool Kit (NLTK) which is used to perform this task provides two main models:

- Maximum Entropy (Maxent)
- Hidden Markov (HMM)

For this work we have used the Maxent approach as it has been trained with a larger corpus and provided better results for our particular scenario, as well as being the recommended tagger by the NLTK group [30]. However the Spanish tagging represent a bigger challenge as there were no trained taggers or tagged corpus for Spanish. We selected a portion of the PENN Treebank, which has around 4.5 million of annotated POS words[31], our sample was around 1% which represent around 45,000 words and started to translate our sample and review the annotated POS. Once corpus was suitable for training we created a Trigram Tagger which is based on a second order Markov Models, and according to Brants[32] it is very efficient and performs just as well as other approaches.

### D. *Atomic Review*

Based on IEEE 830-1998 [8] we look for conjunctions words, if any of these are found, a warning is set as the requirement is not atomic.

### E. *Ambiguous Review*

To determine if a requirement is potentially ambiguous, specifically talking about pragmatic ambiguity, which relies on the word itself, we look for lexical groups that are ambiguous by nature; these sets are foreign words, adjectives and adverbs (both on comparatives and superlatives variations). In this case a user-defined dictionary can be used to include project reserved words and avoid the POS tagger to set them as ambiguous

### F. *Completeness Review*

Once each requirement has been tagged the next step is to look for patterns in the requirement statement that lead to identify the required elements, these patterns need to match our controlled defined grammar which is based on regular

expressions. In Table 1, we provide a subset of the *Penn Treebank Tagset* [26] that is commonly used on our grammar

TABLE I. PENN TREEBANK TAGSET

| POS Tag | Meaning |
|---|---|
| DT | determiner |
| NN | noun, singular or mass |
| MD | modal |
| VB | verb, base form |
| IN | preposition |
| NNP | proper noun, singular |
| NNS | noun, plural |

To evaluate the completeness a grammar based chunk parser is utilized to match the requirement statement grammar with the regular expressions, if all three elements can be identified, the requirement is tagged as complete, otherwise the proper warning flag is set.

Finally, all this theory is implemented in a tool called NLARE which is based on previous works [24].

## V. TESTS AND RESULTS

As mentioned before, to evaluate our proposal we developed a tool called NLARE that automatically applies our approach to review requirements. In this paper we performed the tests with projects from students of the 5th semester in a computer engineering undergraduate program from the University of Baja California. The students worked in a whole development cycle process, starting from requirements up to final release within a time frame of 2 weeks, as part of a class named Object Oriented Advanced Programming.

A total of 60 projects were evaluated, each project were required to have at least 80 functional requirements, generating a total workload of requirements evaluation of 4800. The projects were divided in 2 teams, team A would perform Manual requirement analysis and team B Automatic requirement analysis. Both teams were asked to provide requirement stage in both languages English and Spanish, being the last their native language and English learned by formal and casual methods since university is located in the border with United States.

At the end of project life cycle each project was evaluated for two main attributes:

- Project Quality
- Scheduled Time

Project quality was defined as completion of activities according to documented requirements, how good these activities were implemented, and black box testing results [33]. Scheduled time only reviews if projects were completely finished by the time frame (2 weeks), anything but 100% completed projects were considered not on time. In the following figure we show the average score each team got after project evaluation.

The scores were defined in a 0-100 scale, being 100 a perfect score.

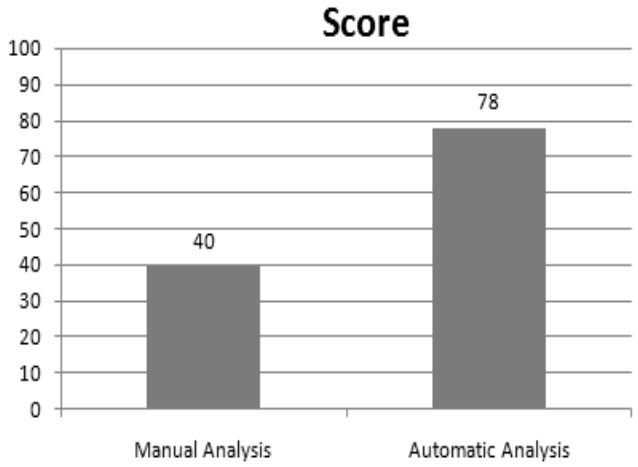


Figure 2. Score comparison for projects.

As we can see in Fig. 2, the projects from students who worked with the automated requirement analysis got a 95% better score then the ones working without requirement stage aids. In the following figure we show the percentage of projects that were released on time for each team.

For the analysis of proper project schedule we can review the following image

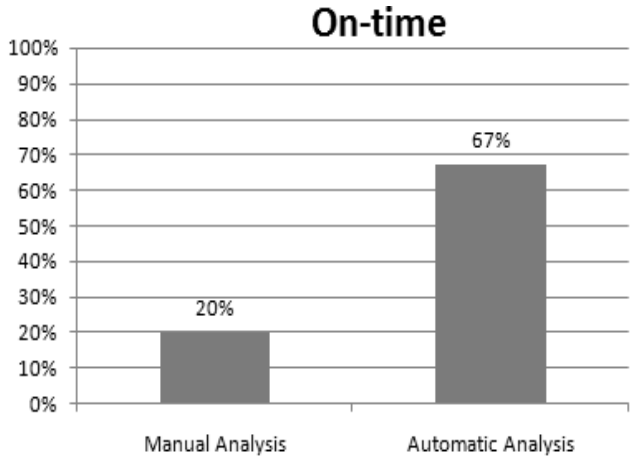


Figure 3. On-time comparison for projects.

From Fig. 3, we can notice that students with aid in requirement stage were able to finish the project on-time

40% more than the other team. As an average, students using the NLARE tool, reviewed requirements more than 2000% faster than manual evaluation, this was according to students clocking an average of ~40 seconds per requirement review as opposed to automatic evaluation of ~58 requirements per second on average laptop computer.

There is of course, a large set of factors that can affect a project success, however, it is possible to trace defects back in the development cycle and find if a requirement was a root cause of a given problematic. It was found that incomplete and ambiguous requirements were the most common defects in requirements and we show the relation with problematic in Table II.

In this case, the incompleteness in the requirements caused delays during the project execution due missing information. In the other hand, also due to the lack of information, the incompleteness originates assuming wrong data that lead to bugs generation or unwanted functionalities. In the case of ambiguity, some misconceived functionalities were implemented wasting efforts while them were rebuilt, besides, ambiguity also generated mismatch between the requirements and design. The team members, in a post-mortem survey, confirmed these consequences.

TABLE II. RELATIONSHIP BETWEEN PROBLEMS AND DEFECTS

| Problem | Cause |
|---|---|
| Delays to look missing information | Incompleteness |
| Wrongly assume information | Incompleteness |
| Wasted efforts on wrong functionalities | Ambiguity |
| Overwork to fix misunderstanding | Ambiguity |
| Mismatch between requirements and design | Ambiguity |

In summary, we could see a considerable improvement for undergraduate student performance when they use formal automated approaches to review their requirements. According to the students post-mortem survey, their lack of vision and experience to write requirements lead to a large amount of modifications in their requirements as they were aware that project quality was going to be evaluated in function of requirements, on the other hand, students using automatic approach commented that even with low experience, the advantage of almost instant requirement evaluation let them create requirements that did not change much over project life cycle, and were easier to develop.

Another evaluated aspect in this work was the performance of the newly introduced support for Spanish language as described in Section IV. The test consist in a comparison of NLARE tool evaluation against manual evaluation to find out how reliable automatic evaluation was, the results are show in Figure 4.

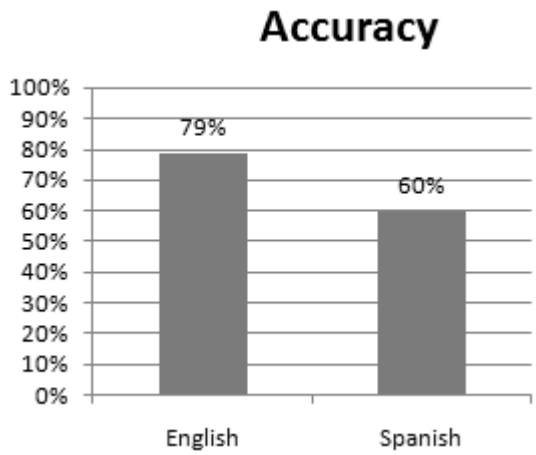


Figure 4. Accuracy comparison between languages

As it can be seen from Figure 4, our current model in English, which currently offers a 78.6% result accuracy as seen in previous works [24] still remains as the most viable solution over Spanish that was only capable of 59.7%. For more in depth analysis of NLARE tool and its performance characteristics, previous works can be reviewed [24].

## VI. THREATS TO VALIDITY

We have found two main potential threats to this proposal, which are related to ambiguity and to correctness of automatic evaluation. We discuss these threats below.

### A. *Ambiguity*

As we can learn from Grenat and Taher studies [27], there are different kinds of ambiguity, listed as: *lexical*, *referential*, *scope* and *structural*. As part of this work, we currently only work with *lexical ambiguity* (often called pragmatic ambiguity), which is ambiguity that lies in the word itself and not the whole context. Given this, a requirement that is affect by one of the other ambiguities will be tagged as correct, however it is the expected result as there is still work to be done in that area.

### B. *Correctness of Evaluation*

In previous works [24] we discuss the overall error detection ratio as well as concordance with human evaluation results for NLARE tool. Results show that our approach gets ~79% match results with correct human evaluation in a fraction of the time. Our research indicates that there is still a possibility that NLARE tool may tag a requirement as correct where this is not, or vice versa. However, our proposal is intended as an aid for requirement engineers but it is not yet a substitute for manual review. This approach will provide a second filter for potential errors to escape to the next stage.

## VII. Conclusions and Future Work

In this paper we have presented a proposal to evaluate the quality of software requirements expressed in natural language. Our proposal is characterized by the effort to formalize the requirements representation and the representation of functional requirements quality problems. In this case, we have considered three of the main problems of natural language requirements: *ambiguity*, *incompleteness*, and *atomicity*. Grammars have been generated to evaluate these quality attributes.

In order to automate the evaluation, we have implemented a prototype tool, which is described in depth in previous works [24]. This tool performs the processing of natural language based on the grammars defined to evaluate the three quality attributes. This process was described in section IV.

The results obtained show that our proposal allows us to gain precision on the quality evaluation, and of course a considerable speedup in the evaluation compared to manual evaluation. Under our testing and since students knew that requirement and design concordance was going to be evaluated, they mention that without the automatic review, a considerable amount of time was used to update or fix wrong requirements. However, these fixes were usually performed after some programming problems were already present, which confirms the studies of Soren[14] that states the sooner the error is detected, the easier and cheaper will be the solution. Even though we have implemented some grammars to attend three specific quality attributes, our model is scalable, this is, we can increase its capacity developing new grammars segments to consider other quality aspects.

The support for Spanish still remains experimental thought, even after our best efforts the limited corpus really hurt our accuracy. Our first attempt to include Spanish language however achieved almost 60% accuracy, we need to get more corpus and test multiple trainers to achieve a higher accuracy, however the evaluation speed still remains as a huge advantage and the automatic evaluation it is intended as an aid for engineers but not a substitute.

As part of the planned future work, we have consider other quality attributes for requirements such as unitary, consistent, traceable, current, important and verifiable, which imply to extend the regular expressions that are the base for our grammar as well as improve our POS tagging step to cover those aspects. Besides, there is plans to focus on improving the Spanish tagging process, this is currently our priority, our plans include to build our own training method as well to be able to match or beat our current benchmark with the English language

## VIII. References


[1] Beck, K. (1999). "Embracing Change with Extreme Programming". Computer 32 (10): 70–77

[2] Boehm, B.; R. Turner (2004). Balancing Agility and Discipline: A Guide for the Perplexed. Boston, MA: Addison-Wesley. pp. 55–57. ISBN 0-321-18612-5

[3] R. Chillarege, W. L. Kao, and R. G. Condit, "Defect Type and its Impact on the Growth Curve," in the 13th IEEE International Conference on Software Engineering, Austin, Texas, May 13-17, 1991.

[4] R. Juárez-Ramírez, "Towards improving user interfaces: a proposal for integrating functionality and usability since early phases", IEEE, Indonesia, 2011.

[5] Brooks, F., "No Silver Bullet: Essence and Accidents of Software Engineering", IEEE Computer, April 1987, pp. 10-19

[6] Alen P Nikora, Experiments in Automated Identification of Ambiguous Natural-Language Requirements, 2011

[7] Davis, Alan M. Software Requirements: Objects, Functions, and States, Second Edition. Prentice Hall. ISBN 0-13-805763-X. 1993

[8] IEEE 830-1998, last reviwed on Feb 03, 2013 at: http://standards.ieee.org/findstds/standard/830-1998.html

[9] S. Boyd, D. Zowhi, and A. Farroukh, "Measuring the expressiveness of a constrained natural languague: an empirical study", Proc. the 13th IEEE International Conference on Requirements Engineering (RE'05), Washington, DC, 2005, pp. 339-352

[10] D.C. Gause and G.M. Weinberg. Exploring requirements: quality before design. Dorset House, New York, 1989

[11] D. M. Berry, E. Kamsties, and M. M. Krieger. From contract drafting to software specification: Linguistic sources of ambiguity, 2003. A Handbook

[12] L. Goldin and D. M. Berry. Abstfinder, a prototype natural languague text abstraction finder for use in requirements elicitation. Automated Software Engineering, October 1997

[13] L. Chung, "On Non-Functional Requirements in Software Engineering", Springer, 2009. p1.

[14] Soren L. & Otto V., "Preventing Requirements Defects", Sixth International Workshop on Requirements, Stockholm, 2000.

[15] Denger C., D. Jorg, E. Kamsties, QUASAR A Survey on Approaches for Writing Precise Natural Language Requirements, 2001, Fraunhofer IESE

[16] S. F. Tjong, Improving the quality of natural language requirements specifications through natural language requirements patterns, March 2006, Technical report, Faculty of Engineering and Computer Sciences, University of Nottingham Malaysia Campus

[17] Wilson W, Rosenberg L, Hyatt L. Automated quality analysis of natural language requirement specifications, in Proceedings of Fourteenth Annual Pacific Northwest Software Quality Conference. 1996.

[18] Fabbrini F., M. Fusani, G. Gnesi and G. Lami, Quality Evolution of Software Requirements Specifications, Proceedings Software and Internet Quality Week Conference, 2000, San Fransisco, CA. pp 3-8.

[19] Götz R. and C. Rupp, Regelwerk Natürlichsprachliche Methode, 1999, Sophist, Nürnberg, Germany

[20] Juristo, N., Moreno, A. M., & López, M. (2000). How to use linguistics instruments for ObjectOriented Analysis. IEEE Software, (May/June): 80–89.

[21] Ohnishi A., Customizable Software Requirements Languages, 1994, Proceedings of the Eighth International Computer Software and Application Conference (COMPSAC), IEEE Computer Society, Los Alamitos, CA

[22] Rolland C. and Proix C., A Natural Language Approach for Requirements Engineering, 1992, pp. 257-277 in Proceedings of Conference on Advanced Informations Systems Engineering, CAiSE, Manchester UK

[23] Halliday, M.A.K. and Matthiessen, C.M.I.M. 2004. An Introduction to Functional Grammar. Arnold: p6.

[24] Carlos Huertas, Reyes Juarez-Ramirez, "A Formal Approach for Measuring the Lexical Ambiguity Degree in Natural Language Requirement Specification", Urke 2011.

[25] Carlos Huertas, Reyes Juarez-Ramirez, "NLARE, A Natural Language Processing Tool for Automatic Requirements Evaluation", Cube 2012.

[26] Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz: Building a Large Annotated Corpus of English: The Penn Treebank, in Computational Linguistics, Volume 19, Number 2 (June 1993), pp. 313--330

[27] Grenat, M., & Taher, M. On the translation of structural ambiguity. Al- Satil (2008), 4, 9-20

[28] Ethnologue: Languages of the World, 17th Edition. (2013)

[29] M. Fusani, S. Gnesi, G. Lami, An Automatic Quality Evaluation for Natural Language Requirements 2001.

[30] Natural Language Tool Kit Initialization module from GitHub. Last checked in April 19/2013.

[31] Marcus M., B. Santoriniy, M. Marcinkiewiczz, Building a Large Annotated Corpus of Spanish: The Penn Treebank. 1993.

[32] Brants, TnT A Statistical Part of Speech Tagger, Sixth Applied Natural Language Processing Conference ANLP-2000. May 2000.

[33] IEEE, "IEEE Standard 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology," 1990.

[34] Mexican American Education Study, Report III, Washington, D.C.: U.S. Government Printing Office, 1972, pp. 76–82