A thorough comparison of NLP tools for requirements quality improvement

Brian Arendse

3657345

b.arendse@students.uu.nl

Master Business Informatics (MBI)

August 2016

1st supervisor: Dr. Fabiano Dalpiaz

2nd supervisor: Garm Lucassen, Msc

Notice of Originality

I declare that this master thesis is my own work and that information derived from published or unpublished work of others has been acknowledged in the text and has been explicitly referred to in the list of references. All citations are in the text between quotation marks (“ ”). I am fully aware that violation of these rules can have severe consequences for my study at Utrecht University.

Date: _____ **Place: _____**

Name: _____ **Signature: _____**

Abstract

One of the main reasons why software projects fail is because of the poor quality of requirements, or the lack of any documented requirements. In the last two decades there has been an increasing interest in the development of tools using Natural Language Processing (NLP) to support the formulation, documentation and verification of NL requirements. Over the years numerous NLP tools have been developed to improve the quality of requirements. Because of the many approaches available it is not clear how the approaches relate to each other. The goal of this thesis is therefore to get a clear overview of the performance of the main approaches taken by NLP tools in the requirements engineering (RE) landscape, and to create a theoretical tool that synergistically integrates the best approaches. The scope is on finding defects and deviations in natural language requirements.

A literature study is performed to identify the main 50 NLP tools in the (RE) landscape. After an initial analysis 3 tools are selected for further analysis. Derived from the features of these 3 tools a requirement standard is created to specify what a quality defect is for each feature. Using the requirement standard 4 datasets are tagged for quality defects. These tagged datasets are compared against the output of the tools using the metrics precision and recall to measure the performance of the features of the 3 tools.

Based on the performance of the features and a qualitative analysis of the approaches of those features a set of good and bad practices is derived:

1. Different tokenizers: The choice to which tokenizer to use can have an effect (both positive and negative) on the performance of a tool
2. Dictionary vs. Parsing: Using a dictionary is a safe and simple method to detect defects. Parsing is a more complicated approaches, and when not performed correctly it can have a negative effect on the performance of a tool
3. What is in the dictionary: The size and content of a dictionary can have an effect on the performance (both recall and precision) of a tool, the bigger the dictionary, the better

The performance of the features and the set of good and bad practices lead to the design of a next generation tool. This tool incorporates the best performing approaches (regarding recall) for each feature specified in the requirement standard. NLP tool developers can use the set of good and bad practices and the design of the next generation tool for the development of their own NLP tools.

Acknowledgment

First and foremost I would like to thank dr. Fabiano Dalpiaz for his active involvement in my master thesis. His guidance, expertise, and patience helped me greatly in writing this thesis. I would also like to thank Garm Lucassen, Msc for providing valuable input to my thesis. Writing the paper would not have been possible without his expertise and know how.

Furthermore I would like to thank Henning Femmer, Msc, prof. Joseph Kasser, and The REUSE Company for providing me with datasets and access to their tools. Where others ignored me, they didn't.

Finally I would like to thank my friends and family for supporting me not only during my thesis, but during the entire Master Business Informatics.

Table of Contents

List of Figures

List of Tables

1 Introduction

The requirements engineering (RE) phase is extremely important in any software project (Kof, 2005). At the end of the RE phase a set of stable requirements is established, after which the system can be designed, developed and implemented. Nowadays agile methods manage requirements throughout the design process, where requirements change all the time (Dingsøyr, Nerur, Balijepally, & Moe, 2012; Paetsch, Eberlein, & Maurer, 2003). Traditional RE and agile development have several differences, but they at least have one thing in common; they need to formulate and document the requirements (Paetsch et al., 2003).

One of the main reasons however why software projects fail is because of the poor quality of requirements, or the lack of any documented requirements (de Bruijn & Dekkers, 2010). Low quality requirements are requirements that are not understood in the same way by different stakeholders (either due to jargon, words with different meanings, fragmented sentences, etc.). Despite the availability of modelling languages and techniques, most of the requirements are still largely written in natural language (NL) so that they can be easily understood by all the stakeholders involved in the project (Berry, Gacitua, Sawyer, & Tjong, 2012; Mich, Franch, & Novi Inverardi, 2004). NL also enables the requirements engineer to specify any type of requirement and as abstract or detailed as required (Kamsties & Peach, 2000). The problem with using NL to specify requirements is that they can be ambiguous and misunderstood (Kof, 2005). Other issues regarding requirements in general is that they can be incomplete, inconsistent, incorrect, infeasible, unusable, and not verifiable (Firesmith, 2003).

Enforcing higher quality requirements can be done in any phase of the RE process. For instance, during the requirements specification phase, a style or structure can be used to specify the requirements (Jain, Verma, Kass, & Vasquez, 2009). Another example is the use of quality models during the requirements verification phase. According to Kamsties & Peach (2000) the previously mentioned solutions for higher quality requirements suffer from either lack of acceptance by the intended users, lack of specificity, or uni-dimensionality, meaning that only the linguistic dimension is addressed and not the pragmatic dimension.

In the last two decades there has been an increasing interest in the development of tools to support the formulation, documentation and verification of NL requirements (D. Berry et al., 2012). There are numerous tools that use natural language processing (NLP) to mitigate the problems and increase the quality of NL requirements. In this thesis these kind of tools are referred to as NLP tools. Liddy (2001) defines NLP as follows:

> *"Natural Language Processing is a theoretically motivated range of computational techniques for analysing and representing naturally occurring texts at one or more levels of linguistic analysis for the purpose of achieving human-like language processing for a range of tasks or applications."*

According to this definition the goal of NLP tools is to “achieve human-like language processing”. When applied to RE this is not exactly the case. Ryan (1993) states that even if a NLP tool is able to understand NL, not all aspects of a system can be explained in NL. NLP tools will never be able to cover the entire requirements engineering process due to its complex nature, and therefore will never be able to achieve human-like language processing. For this reason several NLP tools state that their goal is not to replace the requirements engineer but to support the requirements engineer in the RE process (Ambriola, 2006; MacDonell, Min, & Connor, 2005). The NLP tools automate parts of the RE tasks in the RE process, so that the requirements engineer has to do less work manually. The goal of RE can thus be seen as to aid the requirements engineer in the RE process through using NLP techniques. These techniques can be based on the way humans process NL.

1.1 Problem Statement

NLP has been studied since the 1950s, emerging from the fields of linguistics and artificial intelligence (Nadkarni, Ohno-Machado, & Chapman, 2011). Early approaches focused on production rules and the grammar of a language (Cambria & White, 2014), with the work of Chomsky (1956) being a major influence. In the 1960s the focus changed from a more syntactic approach to semantic pattern matching. A decade later the First Order Logic (FOL) became popular and can be seen as a combination of syntactic and semantic approaches. FOL uses axioms and rules of inferences to formulate relations between concepts by the means of predicates and quantification. The network approach got attention in the 1980s. Examples of the studied networks are Bayesian networks and Semantic networks. According to Liddy (2001) it was not until the 1990’s that the field of NLP gained prominence. She argues that this growth in interest can be explained by the availability of electronic text, the availability of computers, and the rise of the Internet. In 2004 McGuinness & Van Harmelen (2004) introduced the Ontology Web Language (OWL); this language and the related technology are still exploited by some NLP tools.

Numerous NLP tools have been developed over the years. Recent tools use one or more of the previously mentioned approaches and techniques. There are various tools that employ a hybrid approach, using for instance both FOL and OWL (Vincenzo Ambriola, 2006; Jain et al., 2009). Even though there are many approaches, some basic NLP techniques are present in most NLP tools. For instance tokenization and Part-Of-Speech (POS) tagging are NLP techniques that can be found in most of the recently developed NLP tools. Some NLP tools use an external parser for this (e.g. Standford Parser, Minipar, and Treetagger), while other NLP tools developed their own parser. These parsers differ in the way they process NL and in the output they return (e.g. parse tree vs. ER diagram) (Liddy, 2001).

These differences imply is that there is no single approach or technique that can be seen as the golden standard for NLP tools in RE. There have been done comparisons of sub-parts of NLP tools (e.g. POS tagging comparison), showing the performance of individual activities within NLP tools. This has not been done yet, however, for NLP tools and their approaches.

The **goal of this thesis** is therefore to get a clear overview of the performance of the main approaches taken by NLP tools in the RE landscape, and to create a theoretical tool that synergistically integrates the best approaches.

1.2 Scope

The main scope of this thesis is on NLP tools within the RE domain. There are various tasks in the RE process that can be automated by NLP tools. Berry et al. (2012) specifies four broad categories were NLP tools can fall into:

1. Finding defects and deviations in NL requirements documents
2. Generating models from NL descriptions
3. Inferring trace links between NL descriptions
4. Identifying the key abstractions from NL documents

The scope of this thesis is on **finding defects and deviations in NL requirement documents**, and by doing so improving the quality of the requirement documents. The reason why this thesis is limited to the quality of requirement is that this is a main reason why software projects fail or not succeed (de Bruijn & Dekkers, 2010). Additionally, comparing NLP tools with NL requirements document as output against for instance NLP tools with a model as output is not useful. These NLP tools have different goals (quality improvement vs. visualisation), which makes comparing them according to a shared set of metrics difficult. Only sub-parts that overlap with the other type of NLP tools can be compared, which contradicts the goal of this thesis. By comparing only NLP tools that focus on the quality of requirement documents, the approaches of the NLP tools can be compared instead of just sub-parts of the NLP tools.

1.3 Scientific Relevance

According to Cambria & White (2014) future NLP research and development will focus more on the semantics of NL instead of the syntax of NL, on which research has mostly focused ever since the 1950's. They describe that the approaches can be seen as curves which are somewhat sequential (Figure 1). This means that the experiences and knowledge from the syntactic curve are used in the semantic curve. The same goes for the transition from the semantic curve to the pragmatic curve. In current NLP research the syntactic approach is the most popular approach to process NL.

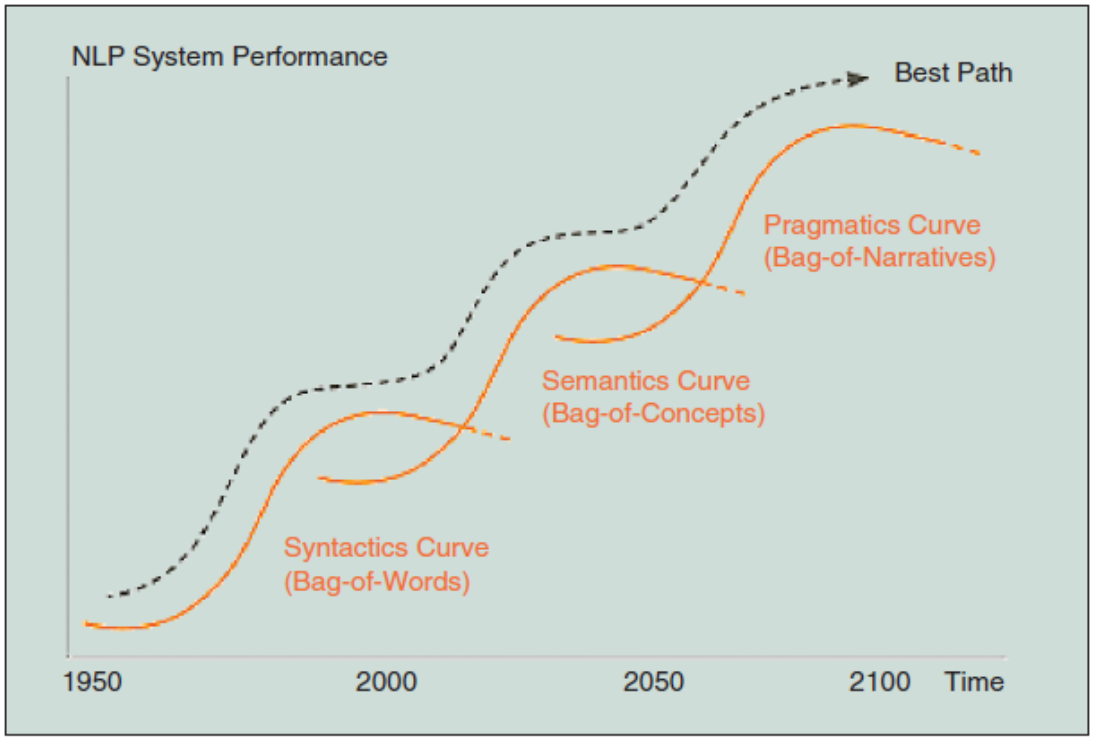

Figure 1 NLP curves

Recently, however,we have been witnessing a shift towards a more semantic approach (Cambria & White, 2014). There are numerous NLP tools designed by following the semantic approach (Kiyavitskaya, Zeni, Mich, & Berry, 2008; Yang, Roeck, Gervasi, Willis, & Nuseibeh, 2012). The increasing popularity of the semantic approach could mean that research is transitioning from the syntactic curve to the semantic curve. Therefore the experiences and knowledge (good and bad practices) of the syntactic curve have to be identified so that research focussing on the semantic curve can progress further.

1.4 Research questions

In order to achieve the goal of this thesis—i.e., to get a clear overview of the main NLP tools in the RE landscape—we define the following research question:

How to create a best of breed Requirements Engineering Tool for improving the quality of requirements through Natural Language Processing?

To answer the main research question, four sub questions are defined. The purpose of the sub questions is to address the information needed to answer the main research question. For each sub question there is a corresponding activity and deliverable. In this section they are briefly mentioned, as they are explained in more detail in the “Research Method” section.

Sub question 1

The first sub question (SQ1) is stated as:

What are the similarities and differences between Natural Language Processing tools?

A literature review is performed to get an overview of the most recent NLP tools. An initial analysis of these NLP tools shows high level differences and similarities between them. From this analysis a selection of the most relevant NLP is made for a more detailed and thorough analysis. This sub question is answered in chapter 4

Sub question 2

The second sub question (SQ2) is stated as:

What are the metrics for analysing and comparing Natural Language Processing tools in terms of quality of requirements?

Another result from the literature review is an overview of methods, frameworks, and metrics that focus on the evaluation and comparison of NLP tools. The next step is to select the methods, frameworks, or metrics that are relevant to this research. This results in evaluation criteria that are used to compare the most relevant NLP tools. This sub question is answered in chapter 4, section 4.3.

Sub question 3

The third sub question (SQ3) is stated as:

What effect does Natural Language Processing tools have on the quality of requirements?

Datasets with requirements serve as input to the most relevant NLP tools. The output is evaluated by means of the selected evaluation criteria. The results are compared together with the thorough analysis of the selected NLP tools. This comparison results in an overview of the strengths and weaknesses of the selected NLP tools (and approaches). This sub question is answered in chapter 6.

Sub question 4

What are good and bad practices of Natural Language Processing tools that can inform the development of future NLP tools for RE?

From the comparison and the strengths and weaknesses of the NLP tools, a set of good and bad practices is derived. This has to serve as a guide or heuristics to both NLP tool developers and requirements engineers. The good and bad practices lead to the design of a conceptual next generation tool. This sub question is answered in chapters 7 and 8.

1.5 Outline

The remainder of this thesis is structured as follows. Chapter 2 describes the research method where the literature study, tools selection, metrics selection, requirements standard, dataset preparation, and experiment design is explained. Chapter 3 describes the related work explaining the theory to

understand this thesis. Chapter 4 provides an overview and analysis of the identified NLP tools. This chapter also explains the metrics used to measure the performance of NLP tools. Chapter 5 describes the requirement standard used in this thesis. Chapter 6 describes how the experiment is conducted and what the results are. Chapter 7 provides a set of good and bad practices based on the results of the experiment. Chapter 8 provides the blueprint for the design of the next generation tool. Finally chapter 9 concludes this thesis and describes the limitations and future work.

2 Research Method

The research method of this thesis is of experimental nature. An input is fed into a subject (in this case a NLP tool), and an output is returned by the subject. The output is evaluated against a pre-defined standard based on the literature and human judgement. The overall research method is depicted in Figure 2.

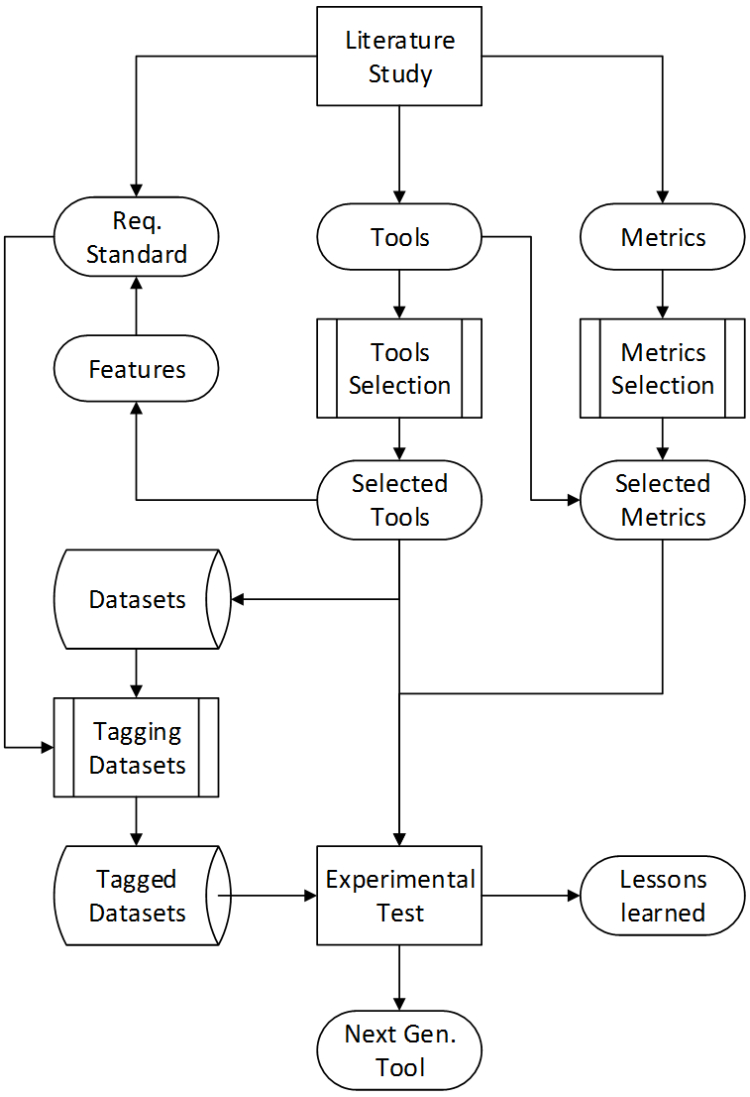

Figure 2 Research Method

At first, a literature study is conducted to identify NLP tools and metrics for the evaluation of NLP tools. Several tools are selected after an initial analysis. The selected tools are analysed more thoroughly. From the available metrics, a subset is selected that is relevant for this research. From the selected tools, a set of features is selected. Each feature is covered by at least one of the selected tools. For every selected feature a definition and guidelines are given, based on the definition of the selected

tools and the literature. The selected features, with the definitions and guidelines, are referred to as the “requirement standard” in this thesis. The next step is to provide datasets containing requirements as input to the selected NLP tools. For every selected tool one dataset is provided by the developers of the tool, of which the developers know that it is processed well by their tool. This dataset serves as a baseline for the tool.

Every dataset is manually processed by the author by the means of the requirement standard, and evaluated by a RE expert. These datasets are referred to as “checked datasets”. All of the datasets are then given as input to each of the selected tools. The output is compared against the checked datasets. The performance is measured using the metrics by looking at the difference between the output and the checked datasets. For every feature the approach of the tool performing best regarding that feature is selected. The result is a conceptual assembled tool that combines the approaches of the selected tools in order to achieve increased performance regarding the quality of requirements. The following sections describe the various parts of the method.

2.1 Literature Study

For the identification of the NLP tools and the evaluation metrics a semi-structured literature review is performed. A semi structured literature review is chosen, and not a structured literature review, because the NLP domain has emerged from the fields of linguistics, computing science, cognitive psychology, and artificial intelligence (Liddy, 2001; Nadkarni et al., 2011). From these domain different concepts are associated and/or used intertwined with NLP. Natural Language Understanding (NLU), Natural Language Generation (NLG), Computational Linguistics (CL), Information Retrieval (IR), and Machine Translation (MT) are some examples of the concepts that are being used intertwined with NLP. Because of the various concepts used in the NLP domain, a structured literature review would result in a large number of search terms required. This, and the large number of papers that need to be reviewed, makes a structured literature review not feasible. The search terms used in this research are therefore limited to the concepts used in the RE domain.

Papers are found using the Google Scholar search engine. The main reason for this is that the relevant libraries such as ACM library, IEEE, and Springer are all queried by the Google Scholar search engine. Most of the found papers are identified using Google Scholar, only a few are found using the dblp computing science library. The literature study consists of three main parts. The first part focuses on the exploration of the NLP and RE domain. The results of this part are described in the chapter “Related Works”. The second part focuses on the identification of the NLP tools. The third part focuses on the identification of the evaluation metrics.

2.1.1 NLP tools

The first step in identifying NLP tools is to formulate a search protocol while doing the literature review. (Yue, Briand, & Labiche, 2011) did a systematic literature review for NLP tools that create models from requirement documents. They came up with three sets of terms to search for available literature; population terms, intervention terms, and outcome terms. Furthermore they formulated inclusion and

exclusion criteria for the selection of the tools. Based on the work of Yue et al. (2011) a set of primary, secondary, and tertiary search terms have been identified for the scope of this research. Please note that the content of the search terms are different than in the literature review of Yue et al. (2011). An initial set of terms was created from the exploratory literature review. In the process of identifying the NLP tools, additional relevant terms were added to the set. Not all combinations of search terms are queried in the Google Scholar search engine. Instead, combinations of search terms found in papers are used. The final set of terms is defined as:

Primary terms:

Natural language, natural language processing, natural language requirements, natural language understanding, requirements, requirements engineering, ambiguity

Secondary terms:

Model, use case, diagram, requirements specification, requirement document, automated, conflict

Tertiary terms:

Checking, verification, extraction, elicitation, selection, identification, detection, abstraction, generation, analysis, transformation, formalisation, tool, finder, system

Various combinations of primary, secondary and tertiary search terms were used to search for papers describing NLP tools. The primary terms have to make sure the search is within the right domain. The secondary terms refer to the input and output of the NLP tools. This limits the search to the scope of this research. The tertiary terms are various concepts used for the activities performed by NLP tools.

A first inclusion criteria for a paper is that an approach using NLP is described. Another inclusion criteria is that the paper describes a system that uses the proposed approach. The final inclusion criteria is that paper is within the RE domain. The exclusion criteria are:

- Not enough technical information
- Not within the scope (e.g. only POS tagging)
- Doesn’t have NL text or requirements document as input or output
- No tool or application (e.g. only prototype)
- Not relevant anymore / outdated

The search using the search terms and inclusion/exclusion factors resulted in an initial list of papers describing NLP tools. A snowballing approach was used to identify additional NLP tools. Most of the NLP tools described in the papers however referred to NLP tools that were outdated and not relevant anymore. Finally some tools were added by doing unstructured searches and suggestions from researcher in the field of NLP and RE. A final list of fifty NLP have been identified to include in the initial analysis and can be found in Arendse & Lucassen (2016).

2.2 Tools selection

From the fifty identified tools a selection is made to include in further analysis and the experimental test. Only the most relevant NLP tools are selected based on inclusion and exclusion criteria. A NLP tool is included when the goal of the NLP tool is to improve the quality of a requirements document, which is in alignment with the goal of this thesis. NLP tools that focus on ambiguity are also included, since reducing ambiguity is also a form of quality improvement. After this 20 NLP tools are selected from the 50 NLP tools in the initial analysis.

Table 1 Tools that focus on finding defects and deviations

Tool Name	Aim	Input	Transformation approach	Automation	Year	Exclusion
Circe	Quality of requirements	Requirements document	Rule based Ontology based	Semi-automated	2006	Not available
NAI	Ambiguity	Requirements document	Rule based Ontology based	Automated	2010	Not available
QuARS	Quality of requirements	Requirements document	Rule based Ontology based	Automated	2001 2004	Not available
CRF Tool	Uncertainty	Requirements document	Rule based Ontology based	Automated	2012	Not available
AQUSA	Quality of user stories	User stories	Rule based Ontology based	Automated	2015	Input type
T1'	Ambiguity	Requirements document	Rule based	Automated	2008	Not available
RAT	Quality of requirements	Requirements document	Rule based	Automated	2009	Not available
Text2Test	Quality of use cases	Use cases	Unknown	Unknown	-	No information
MaramaAI	Quality of requirements	Requirements document	Rule based Pattern based	Semi-automated	2011	Not automated
EuRailCheck	Quality of requirements	Requirements document	Rule based Ontology based	Semi-automated	2012	Not automated
UIMA	Use case model	Use case description	Rule based	Automated	2009	Out of scope
DODT	Quality of requirements	Requirements document	Rule based Ontology based	Semi-automated	2011	Not automated
SREE	Ambiguity	Requirements document	Ontology based	Semi-automated	2013	Not available
Extraction of OLAP req.	Quality of requirements	Requirements document	Rule based Pattern based	Automated	2009	Out of scope
HEJF	Quality of requirements	Requirements document	Rule based	Automated	2014	Included
Dowser	Ambiguity	Requirements document	Rule based Ontology based	Semi-automated	2008	Not automated
RQA	Quality of requirements	Requirements document	Rule based Ontology based		2011	Included
Anaphora detection	Ambiguity	Requirements document	Rule based	Automated	2011	Not available
Lexior	Quality of requirements	Requirements document	Unknown	Unknown		No information Not available
Tiger Pro	Quality of requirements	Requirements document	Ontology based	Automated	2004	Included

Duplicate approaches are excluded from this research. A second exclusion factor is that NLP tools are too domain specific or too high level. For instance UIMA (Ferrucci & Lally, 2004) is a platform where various parsers, annotators, tokenizers can be chosen to create a NLP tool. Even though this would be interesting to research, it does not fall within the scope of this research mainly because our goal is on what the tools do with the information provided by these parsers, annotators, and tokenizers. Thirdly NLP tools that are no longer supported by the developers are excluded from the research. Finally NLP tools that have a semi-automated approach are excluded for further research, tools that require a glossary as supplement are not excluded. The reason for this is that semi-automated NLP tools require a requirements supplement in order to function. This means they require a domain specific ontology, dictionary or vocabulary. It could also mean that the requirements engineer has to do some manual procedures before the tool can function, for instance linking requirements or creating dependencies between requirements. The scope of this research is on automated NLP tools. The 20 tools that focus on finding defect and deviations are listed in Table 1, together with the reason for exclusion.

The exclusion criteria result in 4 NLP tools that are finally included in the thorough analysis and experimental test. After the thorough analysis AQUSA (Lucassen, Dalpiaz, Brinkkemper, & van der Werf, 2015) is excluded because it requires a specific kind of requirements document as input (user stories). Efforts could have been made so that the tool could be involved in the experiment. The approaches of the tool however are already covered by the other tools involved in the experiment. The dataset provided by the developer of this tool is still used in the experiment.

The NLP tools that are included in the experimental test can found in Table 2. The 3 NLP tools that are included in the experimental test will be described and analysed further in section 4.2.

Table 2 NLP tools included in this research

Tool Name	Year of Publication	Reference
Qualicen[1]	2014	(Femmer, Fernández, et al., 2014)
RQA	2011	(Génova et al., 2011)
TIGER-PRO	2004	(Kasser, 2003)

2.3 Metrics selection

For the metric selection an unstructured literature study is performed. The literature search is focussed on frameworks or metrics that can measure the performance of NLP tools, and in particular within the domain of RE. The papers of the identified NLP tools are also analysed to see how the performance of the NLP tools is evaluated. These type of metrics will be referred to as “performance metrics”.

[1] Qualicen is formerly known as HEJF.

2.4 Requirement standard

The requirement standard consists of features that are covered by the selected NLP tools. Only the features within the scope of this research are selected. This consists mostly of features that deal with atomicity and ambiguity. Features that deal with for example model consistency or uniformity have not been included in the requirement standard. Features from different tools that are the same or similar have been grouped into one feature in the requirement standard. Figure 3 shows how the requirement standard has been created.

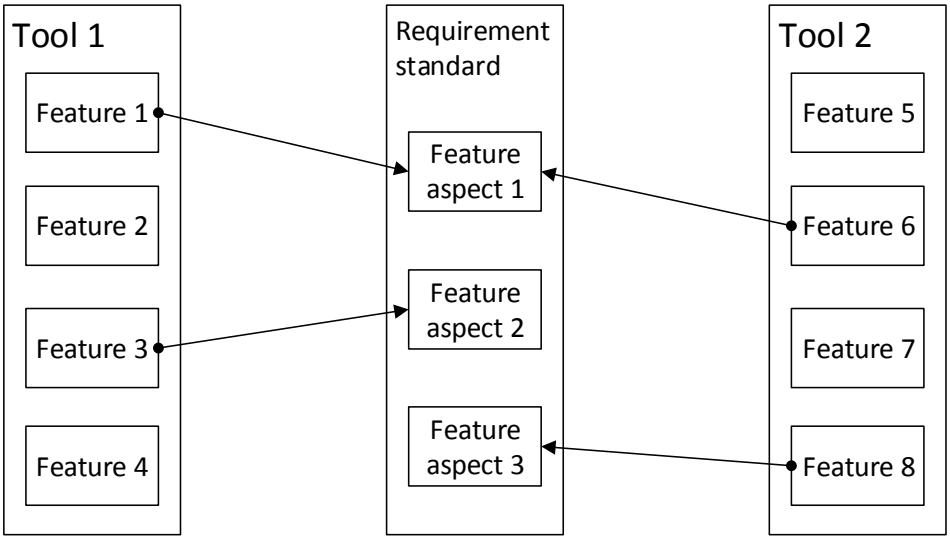

Figure 3 Requirement standard creation

For each feature in the requirement standard a definition and guidelines are given. Every feature has to be specified so that the manual checking of the datasets can be performed in a consistent and repetitive manner. The definition and guidelines of a feature are based on the description given by the tools. If the tools have the same definition, and do not contradict each other, that definition is used in the requirement standard. If however the tools do contradict each other, or no definition or guidelines is provided by the tools, the handbook of ambiguity by Berry et al. (2003) is used to provide a definition and guidelines.

2.5 Datasets preparation

For each of the selected tools one dataset is provided by the developers of the tool. Including the tool that was excluded after the initial analysis four datasets were gathered. Every dataset was manually checked by the author using the requirement standard (Arendse & Lucassen, 2016). These “checked datasets” were evaluated and validated by a requirements expert from Utrecht University.

2.6 Experiment design

The four datasets are given as input to each of the tools. The output of the tools is compared against the checked datasets. The differences between the output and the checked datasets are used to

calculate the performance metrics. Three rules were used while measuring the performance of the tools:

1. If an instance of a true error is detected, but it is marked as something else (e.g. ambiguity instead of superlative term), the detection of the true error is to be considered as an issue for the labelled metric only for metrics that belong to the same category (e.g. not for multiple requirements instead of ambiguity).
2. If an instance of a true error is detected and marked as multiple errors, the detection of the true error is to be considered as one issue for the labelled metric only for metrics that belong to the same category.
3. If an instance of a true error is detected and marked as multiple errors, the detection of the true error is to be considered as multiple issues only for metrics that do not belong to the same category.

Based on the performance metrics a hypothetical tool is assembled from the tools included in the experiment. For each feature the approach of the best performing tool is selected to be included in the hypothetical tool. The best performing tool is selected by choosing a performance metric, or a combination of performance metrics, and select the tool with the highest score regarding that performance metric.

3 Related work

This chapter describes the field of NLP and in particular within the RE domain. The purpose of this chapter is to elaborate on certain aspects of NLP that are necessary to understand this thesis. The first section focusses on requirements and requirement specification (RS). The second section elaborates on the ambiguity of language, and more specific the ambiguity in requirements. The final section explains the main NLP techniques and approaches relevant to this research.

3.1 Requirements engineering

This section elaborates on the concepts regarding requirements engineering.

3.1.1 Requirements

The purpose of a requirement in general is to make sure that the reader of a requirement shares the mental representation of the writer of the same requirement (Harwell, Aslaksen, Hooks, Mengot, & Ptack, 1993). The purpose of a requirement in software development differs from the general purpose of a requirement. In software development the purpose of a requirement is to specify a single function that a system must be able to perform (Firesmith, 2003). In this research we define a requirement as (Harwell et al., 1993):

"If it mandates that something must be accomplished, transformed, produced, or provided, it is a requirement – period"

The purpose of a requirement in software development describes only a small part of the quality attributes a requirement must have (i.e. cohesiveness). Other quality attributes of a requirement are (Firesmith, 2003; Harwell et al., 1993; Wiegers, 1999):

- Completeness:
  - A requirement should be self-contained. A requirement is complete if it requires no further explanation or clarification. A requirement furthermore contains all the relevant information to understand the requirement.
- Consistency:
  - A requirement should use the same vocabulary. A requirement is externally consistent if it is coherent with the high level goals and requirements of the project and organisation. A Requirement is externally consistent if it does not contradict another requirement semantically. A requirement is internally consistent when all the requirements have a uniform pattern.
- Correctness:
  - A requirement should be semantically and syntactically correct. A requirement is semantically correct if it meets all the needs of the stakeholders, if it is a correct explanation of a business goal, and if it is a correct explanation of high level requirements. A requirement is syntactically correct is it uses the proper format, if there are not spelling mistakes, and if the requirement is grammatically correct.

- Currency:
    - A requirement should specify the current needs of the user. Requirement specifications should be updated when the high level goals of a project or organisation change. An obsolete requirement should be removed from the requirement specification.
- Customer/User Orientation
    - A requirement should not contain jargon that customers or users would not understand. Because the technical background of stakeholders can vary any technical terms should be avoided as much as possible. A requirements should furthermore be phrased in the language of the stakeholders.
- External Observability
    - A requirement should not specify technical details of the system. Only characteristics of the application that are observed when using the application should be specified. A requirement should also not specify the architecture, design, implementation, or testing of the application.
- Feasibility
    - A requirement should be implementable by the developers. A requirement should describe an application that fits within the budget, time schedule, and constraints of the project. A requirement should also cohere to the existing hardware and software in place, and to current state of technological advances.
- Lack of Ambiguity
    - A requirement should not have multiple interpretations. The meaning of a requirement should be objective, not subjective. To avoid ambiguity a requirement should not be vague and has to be precise.
- Mandatory
    - A requirement should be prioritized so that it is clear which requirements are mandatory, which are desired by the stakeholders, and which are constraints.
- Metadata
    - A requirement should have metadata. A requirement should have acceptance criteria so that it is clear when the requirement is met. A requirements should also be allocated to a developer or a team. Other metadata are assumptions, identification, prioritization, rationale, schedule, status, and trace.
- Relevance
    - A requirement should be within the scope of the system. A requirement should specify the behaviour of a system, not a user.
- Usability
    - A requirement should be readable by the customers, managers, software architects, designers, programmers, and testers.
- Validatability
    - A requirement should be able to be tested. It should be possible to check if the requirement actually meets the needs of the stakeholders.
- Traceability

    - A requirement should be traceable. It has to be clear where a requirement came from (e.g. which stakeholder) so that more information could be gathered in case of uncertainty.
- Verifiability
    - A requirement should be able to be verified. The features implemented in the system should be able to be verified by the requirement specification. Each feature of the system should be specified in the requirement specification.

It is however hard to imagine an RS with requirements that show all of these quality attributes. The quality attributes should therefore be seen as guidelines that a RE have to keep in mind while writing requirements (Wiegers, 1999). Another issue is that the quality of a requirement is hard to measure (Harwell et al., 1993). The quality of a requirement can be interpreted in two ways. First the quality is not in requirements themselves, but is determined by the reader of the requirements. This is a more qualitative way of trying to measure the quality of requirements. From the general purpose of a requirement could be argued that if the reader understands and agrees with a requirement, or he thinks he does, the goal of the requirement has been achieved. This however means that the quality of a requirement could depend on the capabilities of the reader.

A more qualitative approach to the quality of a requirement could refer to each or all of the above mentioned quality attributes. To what extend does a requirement capture the needs of a user? Are the requirements within the scope of the project? Are the requirements verifiable? Not only are the answers to these questions hard to quantify, some of them can only be answered after the system has been developed (Harwell et al., 1993). If the quality can only be measured a posteriori, and the quality is hard to quantify, it can be argued that measuring the quality of a requirement is neither necessary nor sufficient (Davis & Zowghi, 2006).

There are however some quality attributes that can be quantified, and have already been implemented in NLP tools. There are for instance tools that try to detect verbs and nouns that are not present in the pre-defined vocabulary (Génova et al., 2011). Other tools focus on detecting technical information in a requirement (Femmer et al., 2014; Kasser, 2003). Therefore in this research the quality of a requirement is measured by looking at a selection of individual quality attributes or features that can be quantified.

3.1.2 Requirement specification

The purpose of a requirement specification (or requirement document) is to communicate the requirements between the stakeholders and developers of a tool or system (Paetsch et al., 2003). It serves as an agreement between the stakeholders and the developers on what the tool or system will be able to do. One definition by (Berezin, 1999) of a requirements document is:

"The requirement document states what the software will do. It does not state how the software will do it."

An RS should not contain technical information on how the system will address the wishes of the users. If anything it should be a translation of the wishes of users into a format the developers of a tool can work with during the development process. An RS however should be more than just a translation of the wishes of the users. An RS is not just a collection of requirement statements (Wiegers, 1999). It is statement on its own, which represents the vision of the developers (Berezin, 1999). The stakeholders can see in the RS if their wishes are being met. Other benefits of an RS are (Berezin, 1999; Paetsch et al., 2003):

- An estimation of resources can be made
- A project plan and time schedule can be made
- There is a quality assurance for testing the system
- Changes can be more easily managed

The quality attributes of RS are similar to those of requirement statements. An RS should be complete, correct, concise, consistent, understandable, feasible, unambiguous, modifiable, and traceable (Paetsch et al., 2003; Wiegers, 1999). Even though the quality attributes of an RS are well defined, the structure of an RS is a more complicated issue. According to the IEEE an RS has to include the following topics:

- Interfaces.
- Functional capabilities.
- Performance levels.
- Data structures and elements.
- Safety.
- Reliability.
- Security and privacy.
- Quality.
- Constraints and limitations.

It is argued by (Berezin, 1999) that the level of detail of an RS should depend on the size of the project and the system to be developed. She argues that only for large systems, systems with many users, and systems that are critical to the business a detailed RS is required. In every other case the project only requires a brief RS. This does not only mean that the level of detail can be lower, but also the number of topics addressed could be less.

The level of detail of an RS can influence the amount of confusion with the reader of the RS. A less detailed RS requires the reader to subjectively fill in the gaps in the RS. The more this phenomenon occurs the more ambiguity can occur in an RS. It is the job of the requirements engineer to determine the level of detail so that the amount of ambiguity is kept as low as possible.

3.1.2.1 Types of requirement specifications

Requirements can be represented as use cases, user stories, and NL requirement documents (Lucassen et al., 2015; Yue et al., 2011). User stories originally had little structure, but currently they follow a

strict template (Lucassen et al., 2015). User stories are written from the user perspective as if a user wrote what he required and why he requires it. Use cases are similar to user stories. One of the main differences however is that use cases are presented as a sequence of steps and alternative steps, rather than a story (Yue et al., 2011). Requirement documents tend to have less structure than user stories and use cases. The requirements are written from the perspective of the requirements engineer as if he describes what the system and user must be able to do. The NL requirements are supported by explanatory text and supporting figures and tables. Most NLP tools cannot process figures, since they focus on NL. It is also important for a NLP tool to differentiate between explanatory text and NL requirements.

No matter which requirement representation is used, 2 ways of writing NL requirements exist. The first is using unstructured NL. This means that the requirements engineer does not have to follow a certain grammar or vocabulary. Secondly there is using structured NL. A pre-defined grammar and vocabulary have to make sure that ambiguity is reduced and the quality of the requirements is improved (Yue et al., 2011). In the case of multiple people adding requirements to the project structured NL make sure that the requirements are consistent.

A pre-defined vocabulary can be domain specific for a project, and is often referred to as a glossary. It is a requirements supplement that is part of the RS. Only nouns and verbs that are in the glossary are allowed to be used while writing the requirements. This has to prevent that in one case the term “system” is used and in another case the term “application” is used. Using multiple terms for the same instance could lead to confusion with the reader of the RS. Another requirements supplement is a definition. A definition is similar to a grammar in the way that they both define the notational short hand for defining requirements. A final requirements supplement is the domain model. A domain model describes the key concepts of a domain and what the relationships between them are (Yue et al., 2011).

3.2 Ambiguity

Ambiguity in an unavoidable issue when dealing with requirements. For every requirement there is always someone who can interpret a requirement differently (Berry et al., 2003). It is extremely difficult for a RE, and also NLP tools, to formulate unambiguous requirements (Ryan, 1993). It could therefore be argued that an unambiguous RS does not exist (Berry et al., 2003). There are however more mature RSs, which are understood in a similar way by the vast majority of the stakeholders.

Even though the term ambiguity is ambiguous on its own (multiple definitions of ambiguity exist)(Berry et al., 2003), most definitions agree on the following definition of ambiguity in the field of RE (Brun, Steinar Saetre, & Gjelsvik, 2009; Chantree, Roeck, Nuseibeh, & Willis, 2006; Gill, Raza, Zaidi, & Kiani, 2014; Kamsties & Peach, 2000; Wiegers, 1999):

“A requirement is ambiguous if it admits multiple interpretations despite the reader’s knowledge of the RE context”

3.2.1 Classification of ambiguity

Ambiguity in NL can represent itself in various forms. There are however four classes of ambiguity; lexical ambiguity, syntactic ambiguity, semantic ambiguity, and pragmatic ambiguity. In the following section a definition and example is given for each type of ambiguity (Berry et al., 2003).

3.2.1.1 Lexical ambiguity

Lexical ambiguity is the most basic form of ambiguity. It occurs when a requirement contains a word that has several meanings. There are two types of lexical ambiguity, homonymy and polysemy. The former can be defined as:

"Different words have the same written and phonetic representation, but unrelated meanings and different etymologies"

An example of homonymy is the word *bark. Bark* could refer to the sound a dog makes (1a), or it could refer to the surface of a tree (1b). It is written and pronounced in the same way, but it has two different meanings.

1. a) All that my dog does all day is bark
   b) You cannot eat the bark of a tree

Polysemy can be defined as:

"A word has several related meanings but one etymology"

An example of polysemy is the word *newspaper.* It could refer for instance to an actual instance of a newspaper (2a), or it could refer to the organisation that makes the newspaper (2b). Even though the word has different meanings, the meanings are related. In order to disambiguate a polysemy a lot of contextual information is required. This makes it hard for NLP tools to address this type of ambiguity (Kamsties & Peach, 2000).

2. a) The sports section is the best part of the newspaper
   b) The newspaper announced that they are hiring new writers

3.2.1.2 Syntactic ambiguity

Syntactic ambiguity occurs when more than one grammatical structure can be given to a sequence of words. There are four types of syntactic ambiguity; analytical, attachment, coordination, and elliptical ambiguity. Analytical ambiguity can be defined as:

"The role of the constituents within a phrase or sentence is ambiguous"

An example of analytical ambiguity is when there are more than one adverbs or adjectives describing a noun. This could cause confusion about what belongs to the noun phrase, and how an adverb or adjective influences the noun phrase. Consider the following phrase:

3. a) The English history student

Could be interpreted as:

3 b) The (English) history student

c) The (English history) student

Attachment ambiguity can be defined as:

"A particular syntactic constituent of a sentence, such as a prepositional phrase or a relative clause, can be legally attached to two parts of a sentence"

An example of attachment ambiguity is when a prepositional phrase can both modify a verb and a noun (or noun phrase) in a sentence. There are two interpretations possible in the example sentence (4a). Either the father provided the children with high grades because of a certain accomplishment. Or the children were rewarded by the father because of their high grades.

4. a) The father rewarded the children with high grades

The third type of syntactical ambiguity is coordination ambiguity. This can be defined as:

"More than one conjunction, and or or, is used in a sentence or one conjunction is used with a modifier"

Even though the definition itself is ambiguous, multiple connectors in a sentence could cause ambiguity. A connector can conjunct two sentences into one sentence. Using multiple connectors, where one connector is not used for a conjunction, can lead to confusion. It is hard for the reader to identify which connector is used for the conjunction of the sentences. In example 5a Susan could be looking at John and Stan, and Steve wasn't looking at John and Stan. Another interpretation is that Susan looked at John, and Stan and Steve weren't looking at John.

5. a) Susan looked at John and Stan and Steve didn't look

Coordination ambiguity can also occur when a single connector is used in combination with a modifier. In example 6a both the table and chair could be green, or just the table could be green.

6. a) Green table and chair

The last type of syntactical ambiguity is elliptical ambiguity, which can be defined as:

"An ellipsis is a gap in a sentence cause by omission of a lexically or syntactically necessary constituent. Elliptical ambiguity occurs when it is not certain whether or not a sentence contains an ellipsis"

In other words, if a word or word group is omitted from a sentence this could cause ambiguity. In example 7a the constituent "knows" after Vincent is left out. If the constituent wasn't omitted the sentence could only be interpreted that Theo knows a taller man than anyone Vincent knows. By

omitting the constituent the sentence could also be interpreted that Theo knows somebody that is taller than Vincent is.

7. a) Theo knows a taller man than Vincent

3.2.1.3 Semantic ambiguity

Semantic ambiguity occurs when there is no lexical and syntactic ambiguity, but a sentence still has more than one interpretation. There are three types of semantic ambiguity; coordination ambiguity, referential ambiguity, and scope ambiguity. Coordination ambiguity is already discussed in the previous section. In some cases of coordination ambiguity the ambiguity is not caused by a single word or group of words. In those cases the coordination ambiguity is classified as semantic ambiguity. Referential ambiguity can be within a sentence or between a sentence and the context. In the former case the referential ambiguity is classified as semantic ambiguity, in the latter case it is classified as being pragmatic ambiguous. Referential ambiguity will be discussed in the next section. The final type of semantic ambiguity is scope ambiguity, which can be defined as:

"Operators can enter into different scoping relations with other sentence constituents"

Quantifier operators are words such as "all", "each", "a", and "every". These operators precede and modify nouns to the scope of the quantifier operator. For instance "all scientists" include every scientist that exists, where "some scientists" only include a selection of the entire population of scientists. Ambiguity can occur when several quantifier operators are included in a sentence. Consider the following phrase:

8. a) All kids like a superhero

If "all" is included in the scope of "a" then there is one superhero that is liked by every kid in the population. If however "a" is included in the scope of "all" then every kid in the population likes a superhero, but not necessarily the same superhero. Besides quantifier operators there are also negation quantifiers such as "not" and "no one". Ambiguity can occur in the same way as with quantifier operators. Consider the following phrase:

9. a) No one has seen a dragon

The sentence could be interpreted that dragons do not exist, and therefore no one could ever see a dragon. Another interpretation is that a dragons does exist, but it hasn't been seen by anyone.

3.2.1.4 Pragmatic ambiguity

Pragmatic ambiguity occurs when multiple meanings can be given to a sentence based on the context of the sentence. The context can be language specific and non-language specific. (Kamsties & Peach, 2000) defines three types of non-language specific context:

- Application domain
- System domain
- Development domain

In the application domain context ambiguity can occur because of what is known about the application. The same goes for the system domain and development domain for what is known about the system and the development process respectively. Based on the context, either one of the three non-language specific context or the language specific context, pragmatic ambiguity can occur. There are two types of pragmatic ambiguity, referential ambiguity and deictic ambiguity. Referential ambiguity can be defined as:

"An anaphor can take its reference from more than one element, each playing the role of the antecedent"

A reference is a word or word group that refers to an actual object in the real world. An anaphor is a word or word group that refers to a part of a sentence, this could be the same sentence or another sentence. The part of the sentence the anaphor refers to is called an antecedent. Referential ambiguity can occur if it is not clear what the antecedent is to which the anaphor refers to. The most common type of anaphor is a pronoun (Berry et al., 2003). Consider the following phrase:

10. a) The kids shall eat the vegetables before they get spoiled

The pronoun "they" could refer to both the kids and the cookies. It is not clear whether the kids get spoiled if the kids don't eat their vegetables, or if the vegetables get spoiled if the kids don't eat the vegetables. Other types of anaphora are definite noun phrases and ellipses. Consider the following phrase:

11. a) If the nickname is available, the user enters the password. If not, the user is rejected

In example 12a it is not clear if the user is rejected if the nickname is not available, or if the user is rejected because he didn't enter a password. A part of the second sentence is omitted which causes an ellipsis between sentences.

3.3 Natural language processing

3.3.1 Levels of language

Before looking at how NLP tools process NL, the way humans process NL has to be understood. This starts by analysing how NL is structured. According to Liddy (2001) there are 7 levels of language that humans use to process NL:

1. Phonology: the interpretation of speech sounds within and across words
2. Morphology: the componential nature of words
3. Lexical: the interpretation of the meaning of individual words
4. Syntactic: the analysis of words in a sentence to uncover the grammatical structure of a sentence
5. Semantic: the determination of possible meanings of a sentence
6. Discourse: the interpretation of a unit of text larger than a single sentence

7. Pragmatic: the interpretation of text by looking at the context rather than the content of the text

In early literature these levels were described as being sequential for processing NL. So before the lexical level of meaning could be reached the phonological and morphological meaning had to be obtained first. Recent research suggests this process is much more dynamic and can be seen as a synchronic model (Liddy, 2001). This means that humans give meaning to NL on every level of language. Humans also use information from a "higher" level to help provide meaning to a "lower" level of language. For instance pragmatic information such a knowing the domain could help provide meaning to individual words on the lexical level of language.

NLP tools can focus on each level of language, although in RE the phonological level is rarely addressed. Humans use every level to understand NL. In order for a NLP tool to even come close to human NLP Liddy (2001) states that:

"…the more capable an NLP system is, the more levels of language it will utilize."

In other words this means that there is a correlation between the performance of a NLP tool and the levels of language is uses to provide meaning to NL. This does not mean however that addressing more levels of language results in a better performing NLP tool. What is important is that information from every level of language is shared with other levels in order to help giving meaning to NL. This is however a complicated process and hard to implement into a NLP tool.

3.3.2 NLP activities

NLP can be applied in various domains such as education, finance, and RE. Activities were NLP can be applied include (Ambriola & Gervasi, 1997; Cambria & White, 2014; Liddy, 2001; Mich et al., 2004):

- Information extraction (Xiao, Paradkar, Thummalapenta, & Xie, 2012)
- Information retrieval
- Dialogue systems (e.g. customer services)
- Summarisation
- Machine translation
- Requirements elicitation
- Conflict identification
- Requirements validation
- Requirements selection
- Model creation
- Ambiguity detection (Yang, Roeck, Gervasi, Willis, & Nuseibeh, 2011)

The wide range of activities where NLP can be applied in shows it is a powerful technique that can be implemented in almost every project that uses natural language.

3.3.3 NLP approaches

According to Liddy (2001) a NLP tool falls in 1 of 4 categories; symbolic, statistical, connectionist, and hybrid. Another categorisation is made by (Cambria & White, 2014), who defines 3 categories; syntactic, semantic, and pragmatic. The categorisations made by Liddy (2001) and Cambria & White (2014) have several similarities and differences. The symbolic approach is similar to the semantic approach. They both focus on using lexicons, corpora, and ontologies to provide meaning to NL. The difference is that the semantic approach also includes the connectionist approach in the way that machine learning algorithms are used. Next the statistical approach is similar to the syntactic approach. The syntactic approach however consists of 3 parts; keyword spotting, lexical affinity, and statistical NLP.

Based on this comparison a combination of the categorisations is used for the approaches to NLP. The main categories of Cambria & White (2014) are used since they have a broader definition than the categorisation of Liddy (2001). In this research the NLP approaches are defined as:

1. Syntactic: focusses on the analysis of word by using mathematical techniques and corpora without adding any domain knowledge
2. Semantic: focusses on the analysis of concepts and the intrinsic meaning of NL by using ontologies and machine learning techniques
3. Pragmatic: focusses on the analysis of narratives
4. Hybrid: focusses on more than one of the previous approaches.

3.3.4 Pre-processing approaches

Pre-processing is the process of preparing requirements by the means of NLP techniques for the transformation and analysis of requirements. According to Cambria & White (2014) and Liddy (2001) the 5 main pre-processing techniques are:

- Lexical analysis
- Syntactical analysis
- Semantic analysis
- Categorisation
- Pragmatic analysis

The pre-processing techniques are similar, to some extent, to the levels of language. As described earlier better performing NLP tools use more levels of language, and thus use more pre-processing techniques. In the upcoming section a more detailed description of the pre-processing techniques is provided.

3.3.4.1 Lexical analysis

The first step in processing NL is performing a lexical analysis. The purpose of the lexical analysis is to interpret the meaning of individual words (Liddy, 2001). Lexical analysis can be defined as (Dale, Moisl, & Somers, 2000):

"The determination of lexical features for each of the individual words of a text"

The extent to which a NLP tool performs a lexical analysis is based on the complexity of the tool (Dale et al., 2000; Liddy, 2001). There are four main techniques used in the lexical analysis; sentence splitting, tokenization, Part-Of-Speech tagging, and morphological analysis. The techniques are listed and described from the most basic to the most complex technique.

Sentence splitting

Sentence splitting is also known as sentence boundary detection or sentence segmentation. The goal is to split the text into sentences so that these can be processed further (Dale et al., 2000). During sentence splitting the NL text is analysed to determine the sentence boundaries between the sentences. Most languages use punctuation marks to indicate the boundaries between sentences. There are however some instances were punctuation marks are not used for indicating boundaries. For instance with abbreviations and titles punctuation marks are used, which do not indicate a sentence boundary (Nadkarni et al., 2011). Information from the tokenization process is therefore required to help with the detection of sentence boundaries. The process is further complicated when an abbreviation is at the end of a sentence, and the period marks both the abbreviation and the end of the sentence (Dale et al., 2000).

Tokenization

Tokenization is also known as word segmentation. The goal is to split the text into elements, called tokens, so that these can be processed further (Dale et al., 2000). Based on the structure of the text, which is partly provided by the sentence splitting, the tokens are associated to a category. The most common categories are words (consisting of letters only), numbers, punctuation marks, and symbols (Ferilli & Singh, 2011). A more basic method of tokenization is looking up the tokens in a table to determine the category it belongs to. This approach is however mostly used for simple tasks (Dale et al., 2000).

Part-Of-Speech tagging

The purpose of Part-Of-Speech (POS) tagging is to assign the words of a text into word classes (Weikum, 2002). Voutilainen (2003) defines eight classes of words; noun, verb, particle, article, pronoun, preposition, adverb, and adjective. This classification is supported by Weikum (2002), with a difference in the terminology of the classes. Instead of an article, Weikum (2002) defines a determiner, which is the superclass of an article. Information from the tokenization process is used to classify the words. An example output of POS tagging could like:

12. a) A(Art) cat(N) has(V) 9(Nu) lives(N)

Where Art is an article, N is a noun, V is verb, and Nu is a numeral. The noun "lives" is an example of what makes POS tagging challenging. When only looking at individual words it could also be classified as a verb. Therefore information from the morphological analysis is used to assign words to their word classes.

Morphological analysis

Morphological analysis is also known as morphological decomposition and morphological normalization. The purpose of this step is to identify the root of compound words (Nadkarni et al., 2011). This can be accomplished by using stemming, lemmatization, and suffix stripping. With stemming and lemmatization a dictionary is involved. With stemming a word such as 'computational' is brought back to the root 'comput', and with lemmatization the root would be 'compute'. Suffix stripping does not require a dictionary, but removes standard suffixes that follow the root of a word.

3.3.4.2 Syntactic analysis

The output of the lexical analysis serves as input to the syntactic analysis. The goal of the syntactic analysis is to uncover the grammatical structure of a sentence (Liddy, 2001). The information of the grammatical structure can be used, in combination with rules, to identify quality defects.

3.3.4.3 Semantic analysis

The purpose of the semantic analysis is to determine possible meanings of a sentence (Liddy, 2001). All the information from the previous analysis is used for this. Semantic analysis looks at the meanings of the different words in a sentence. Cambria & White (2014) define 2 broad types of semantic analysis for NLP: endogenous NLP and taxonomic NLP. Endogenous NLP deals with machine learning techniques, where taxonomic NLP focusses on dictionaries, ontologies, and corpus.

3.3.4.4 Categorisation

The purpose of text categorisation is to classify certain aspects of text (Weikum, 2002). Categorising in requirements engineering deals with classifying requirements for a specific purpose. Classifying requirements can be useful for the development of software. Classified requirements can be assigned to teams that each focus on a particular class of requirements (Yue et al., 2011).

3.3.4.5 Pragmatic analysis

The output of the previous analysis only identifies the defects in the requirements. The pragmatic analysis automatically improves the requirements based on the identified defects in the previous analysis (Yue et al., 2011). Another form of pragmatic analysis is that improved requirements, based on identified defect earlier, are checked whether the defects are still present.

3.3.5 Transformation approaches

After pre-processing the requirements they are transformed into an analysis model, intermediate model or improved requirements. The latter does not necessarily mean the generation of improved requirements, it could also be in the form of error messages and warnings. In the remainder of this section the 3 types of transformation approaches are described.

The first type of transformation approach is the rule based approach. For this approach a set of pre-defined transformation rules is used. These transformation rules compare the input requirements with heuristics, definitions, boilerplate requirements and other requirement supplements that provide a

structure or template. The rule based transformation approach is often seen in NLP tools that follow the syntactic NLP approach.

The second type of transformation approach is the ontology based approach. For this approach a vocabulary or ontology is used. An ontology “defines the terms used to describe and represent an area of knowledge” (Heflin, 2004). This representation is in a form that can be processed by computers and is unambiguous (Yue et al., 2011). A simple example of an ontology is a taxonomy describing the relevant terms within a domain and the relationships among them. Ontology Web Language (OWL) is a technique that can be used to create an intermediate model from requirements by creating an ontology model. This model can be a class diagram consisting of classes and attributes and the between them (Heflin, 2004).

The third type of transformation approach is the pattern based approach. For this approach a target pattern is required. Requirements that have a certain pattern are thus transformed into requirements that another, target, pattern.

4 NLP tools

One part of the literature study consists of getting an overview of the most recent NLP tools. In this chapter the results of this part of the literature study are described. The first part of this chapter focusses on all of the NLP tools that are found in literature. From these tools a selection is made for further analysis. The selected NLP tools are analysed in detail in the second part of this chapter. The third part discusses the metrics that can be used to measure the performance of NLP tools.

4.1 NLP tools overview

From the literature study a set of 50 NLP tools are identified relevant to this research. An initial analysis is performed on these 50 tools ((Arendse & Lucassen, 2016)). At first the goal, input and output of each tool is identified to determine whether the tool was within the scope of this research. The scope of this research is on NLP tools that focus on improving the quality of requirements and focus on ambiguity. Within the RE domain however there are various other goals that NLP tools can focus on (see Figure 4).

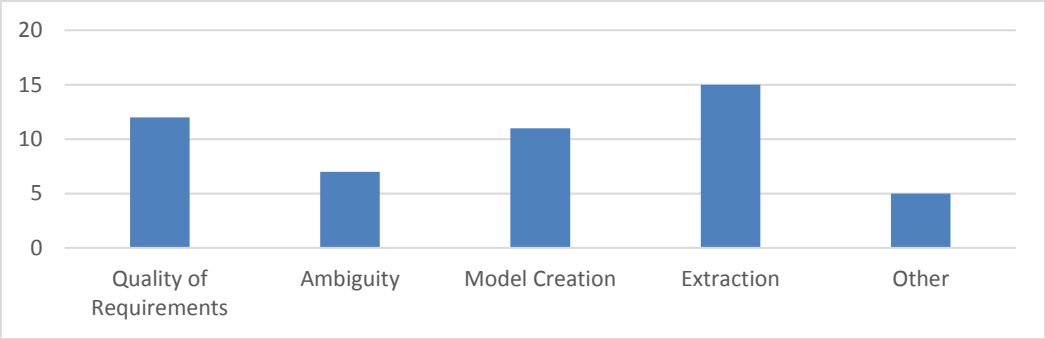

Figure 4 NLP tools distribution

Besides the quality of requirements and ambiguity there are 2 other main goals of NLP tools. The first is model creation and refers to tools that create various models such as UML model, BPMN model, and feature model from a RS. The second is extraction and refers to tools that extract glossaries, relevant terms, requirements, and business rules from NL text. The tools in the "Other" category are tools that either only use NLP in a part of their approach or are part of a larger approach (e.g. NLTK is used in AQUSA for the lexical analysis).

The input of the tools are forms of NL text. This can be user stories, use cases, NL requirements, NL requirement documents, or just plain text. The output of the tools depends both on the goal and the input of a tool. It makes sense that a tool that creates models has a model as output. But for tools that focusses for instance on the quality of requirements the output can differ. There are tools that give some sort of report in various forms such as warnings, errors and defects as output. Other tools however actually give an improved or altered requirement as output when this is given as input.

A second aspect of the initial analysis is to determine whether or not a tool performs a lexical, syntactic, semantic, and / or pragmatic analysis. Furthermore it is analysed if some sort of categorisation is performed by the tools. The various analyses are interpreted differently here than described earlier. The reason for this is that there are not many tools that actually perform a semantic analysis, let alone a pragmatic analysis. By interpreting the types of analyses broader a more detailed analysis can be performed. Therefore the types of analysis are defined as follows:

- Lexical analysis is the pre-processing of NL by the means of tokenisation, sentence splitting, POS tagging and / or morphological analysis
- Syntactic analysis is the process of taking the output of the lexical analysis and give it grammatical structure based on a pre-defined grammar or set of rules
- Semantic analysis is the process of giving some sort of meaning to the output of the syntactic analysis by the means of a corpus, vocabulary, dictionary, or other methods
- Categorisation is the process of automatically ordering or classifying requirements for a specific purpose
- Pragmatic analysis is the process where the results of the semantic analysis are used to determine the errors in the input, generate a model, prepare and verify the output, visualize the output, and / or create a report.

When looking at the lexical analysis of the tools it is noticeable that only a few tools have created their own solution for this. Most tools use external software to perform the lexical analysis. The main reason for this is that there are various parsers that prove to have a good performance. The parsing of NL is only mentioned briefly in most of the papers where the tools are described in. There are numerous tools and techniques available that have high precision and recall when it comes to parsing NL. This implies that even for a 'simple' task such as parsing approaches can differ. Some parsers even cover both the lexical and syntactic analysis. The most commonly used parser is the Stanford Parser. It is often mentioned to be the most validated and accessible parser available. It is also available for multiple languages created by the community of the Stanford Parser. An overview of all the parsers used by the NLP tools included in this research can be found in Table 3.

Table 3 NLP Parsers

TreeTagger	Sent detector	Minipar
Genia Tagger	NLTK	Stanford Parser
OpenNLP Parser	GATE	ANNIE POS tagger
Link Grammar	Moby POS II	SensAgent
RCNL Parser		

The differences between the tools and their corresponding approaches are mostly in the way they perform the semantic and pragmatic analysis. WordNet is a commonly used corpus during the semantic analysis. The way WordNet is used however differs from tool to tool. In the prototype tool described in (Kiyavitskaya et al., 2008) wordNet is used to create a Semantic Net. In the tool (Ibrahim & Ahmad, 2010) wordNet is used as part of a concepts extraction engine. What this shows is that a

corpus or dictionary can be used in various ways to give meaning to text or parts of text. Besides the use of a corpus there are various other methods that are being used to give some sort of meaning to text. Some tools create an intermediate model to represent the text in a format where relationships and dependencies can be constructed between parts of text. Some of the models that are being created are world model, framenet, and essential use case model.

During the pragmatic analysis missing relationships and dependencies are detected by using rules and pre-defined model formats. The feedback is given to the user by providing error messages or warnings. Other methods used during the semantic analysis include named entity recognition, conditional random fields, semantic role labelling, annotation schema, attribute value splitting, semantic pattern matching, and two level grammar.

Another aspect where NLP tools can differ is in the requirements supplements they require in order to function. As mentioned earlier some tools require a corpus or dictionary to give meaning to the text. These tools are mostly ontology based. Other tools are more rule based and require rules, definitions, heuristics, and sometimes a glossary as a requirements supplement. The output of the syntactic analysis, mostly a parse tree, is analysed by the means of the requirements supplements. Tools that require a glossary as supplement check the nouns of the text with the glossary. This way they check whether all the nouns are specified in the glossary in order to get term consistency within the RS. A glossary differs from a corpus or dictionary in the way that a glossary is a list of “allowed” terms and a corpus is a list of “disallowed” terms.

For a lot of the NLP tools the requirements supplements have an impact the automation of the tool. Some tools require a domain model or domain dictionary in order to function properly. In most cases this results in a tool that is semi-automated. These tools mention however that domain specific information can increase the performance of the tool. Automated tools on the other side are mostly domain independent and require minimal manual work before the tool can be used.

4.2 NLP tools in experiment

From the 50 identified tools, 20 tools focus on finding defect and deviations. By using exclusion criteria, as described in section 2.2, 3 tools are involved in the experiment (RQA, Qualicen, and TIGER-PRO). This section describes each of the 3 tools, and the approaches they use.

4.2.1 RQA

The Requirements Quality Analyzer (RQA) is a commercial tool developed by The REUSE Company. RQA can be implemented in larger organisations where multiple people work on the same project. The tool offers metrics covering the correctness, consistency, and completeness of requirements. For this they define 4 types of indicators (Génova et al., 2011):

1. Morphological indicators, such as size
2. Lexical indicators, such as number of ambiguous terms
3. Analytical indicators, such as usage of verbal forms

4. Relational indicators, such as overlapping with other requirements

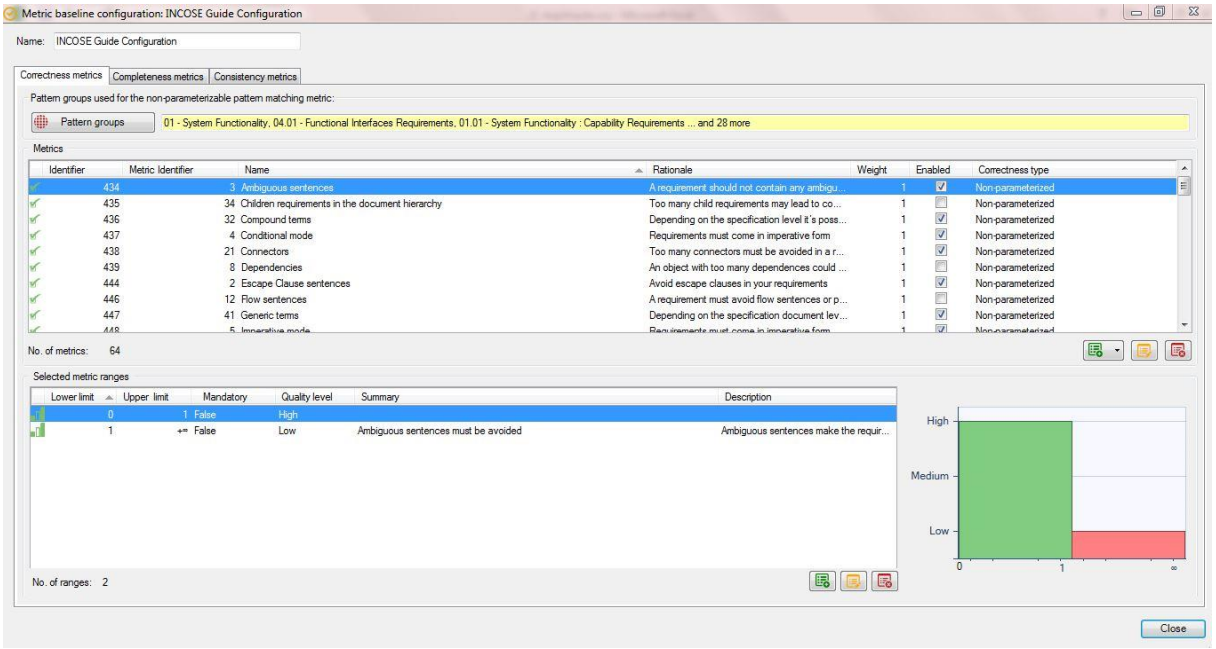

Figure 5 RQA metrics

Only the morphological and lexical indicators are relevant for this thesis, because they cover the correctness of requirements. The tool offers metrics that can be adjusted by the user. For instance the occurrence of an ambiguous term weighs twice as much as the occurrence of a connector. Additionally the user can modify the dictionaries RQA uses (Figure 5). The user can define nouns, verbs, adjectives, and adverbs that are considered worth looking for by the tool. The analytical indicators help the user in using a pre-defined pattern for requirements. The relational indicators provide a similarity measure that tells the user how much the requirements overlap.

After RQA performs the analysis on the requirements it provides the user with errors and warning. The tool gives a score for each metric and a short explanation what the metric means (Figure 6). The words highlights red when there is an error message regarding those words. The user can export the results of the analysis to a pdf or excel file.

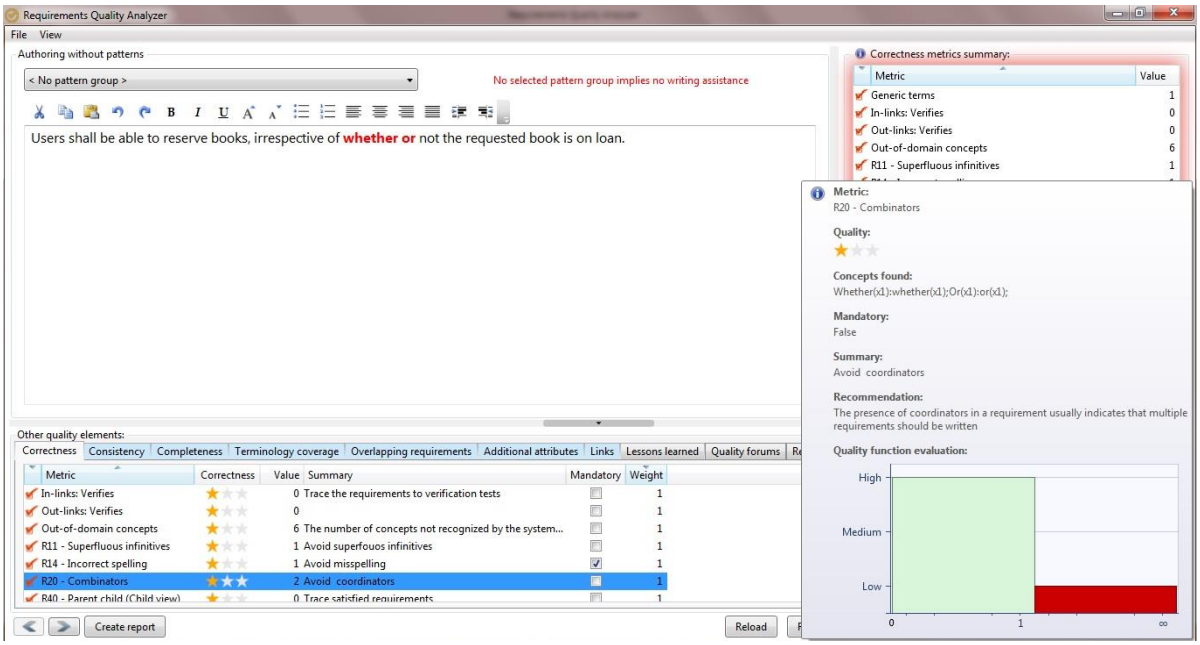

Figure 6 RQA report

4.2.2 Qualicen

Qualicen is a commercial tool that stemmed from a research initiative at TU Munich. The tool analysis NL requirements by using requirement smells. Qualicen defines a requirement smell as "a concrete instance for a requirement artefact's quality defect" (Femmer et al., 2014). The difference between a requirement smell and a requirement defect is that a smell is only an indication for a possible quality defect. Qualicen detects the following requirement smells:

- Slash smell
- Ambiguous adverbs and adjectives smell
- Negative words smell
- Non-verifiable term smell
- Subjective language smell
- Imprecise phrase smell
- Superlative requirements smell
- Comparative requirements smell
- Vague pronouns smell
- Loophole smell
- UI detail smell
- Long sentence smell

All but the last 3 smells are within the scope of this thesis. The tool provides users with warning messages which a short description whenever a smell is detected (Figure 7). For some of the smells the tool uses dictionaries, which cannot be adjusted by the user. Qualicen however can provide a version of the tool with adjusted dictionaries. Other smells use NLP techniques such as morphological

analysis and POS tagging. In both cases there are rule based exceptions for certain smells that are not quality defects

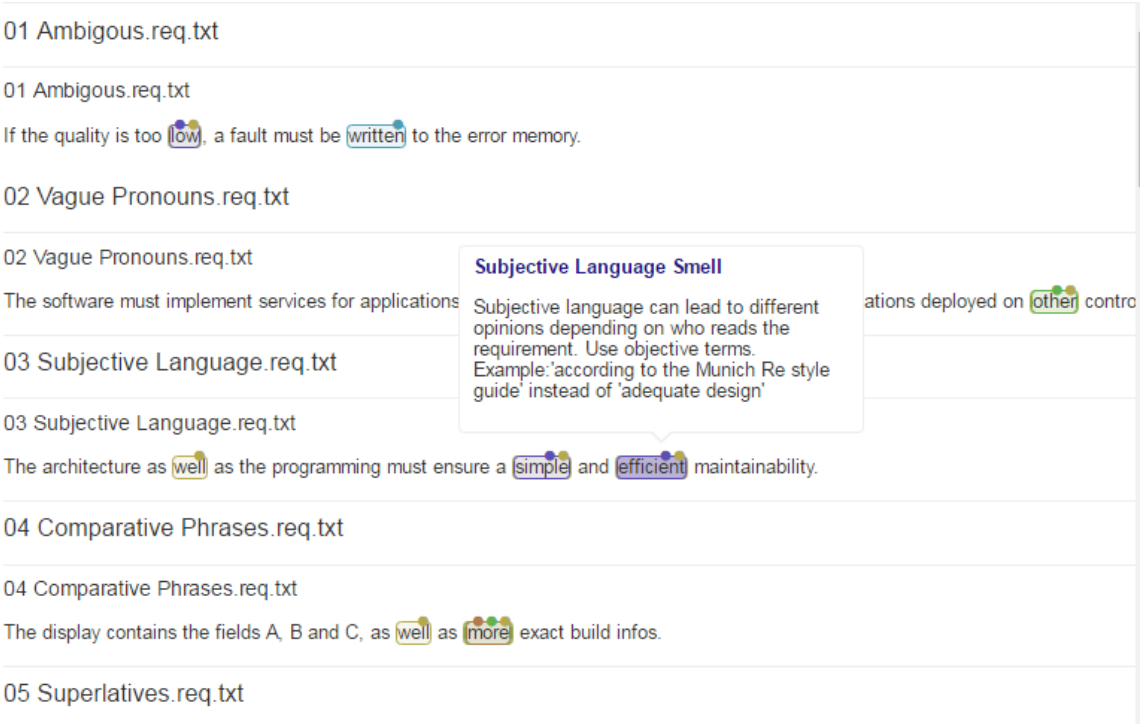

Figure 7 Qualicen report

4.2.3 TIGER-PRO

TIGER-PRO is an educational tool developed by Joseph Kasser. The purpose of TIGER-PRO is to help students write higher quality requirements. The tool defines 7 types of requirement defects:

1. Multiple requirements
2. Possible multiple requirements
3. Unverifiable terms
4. Wrong word
5. User defined words
6. Possible design words
7. Incomplete

All the approaches of TIGER-PRO use a dictionary to check for any defects. The dictionary of unverifiable terms consists of more than just non-verifiable terms. This dictionary also includes quantifiers, subjective terms, comparative terms, slashes, and escape clauses. Figure 8 provides a screenshot of the user interface of TIGER-PRO.

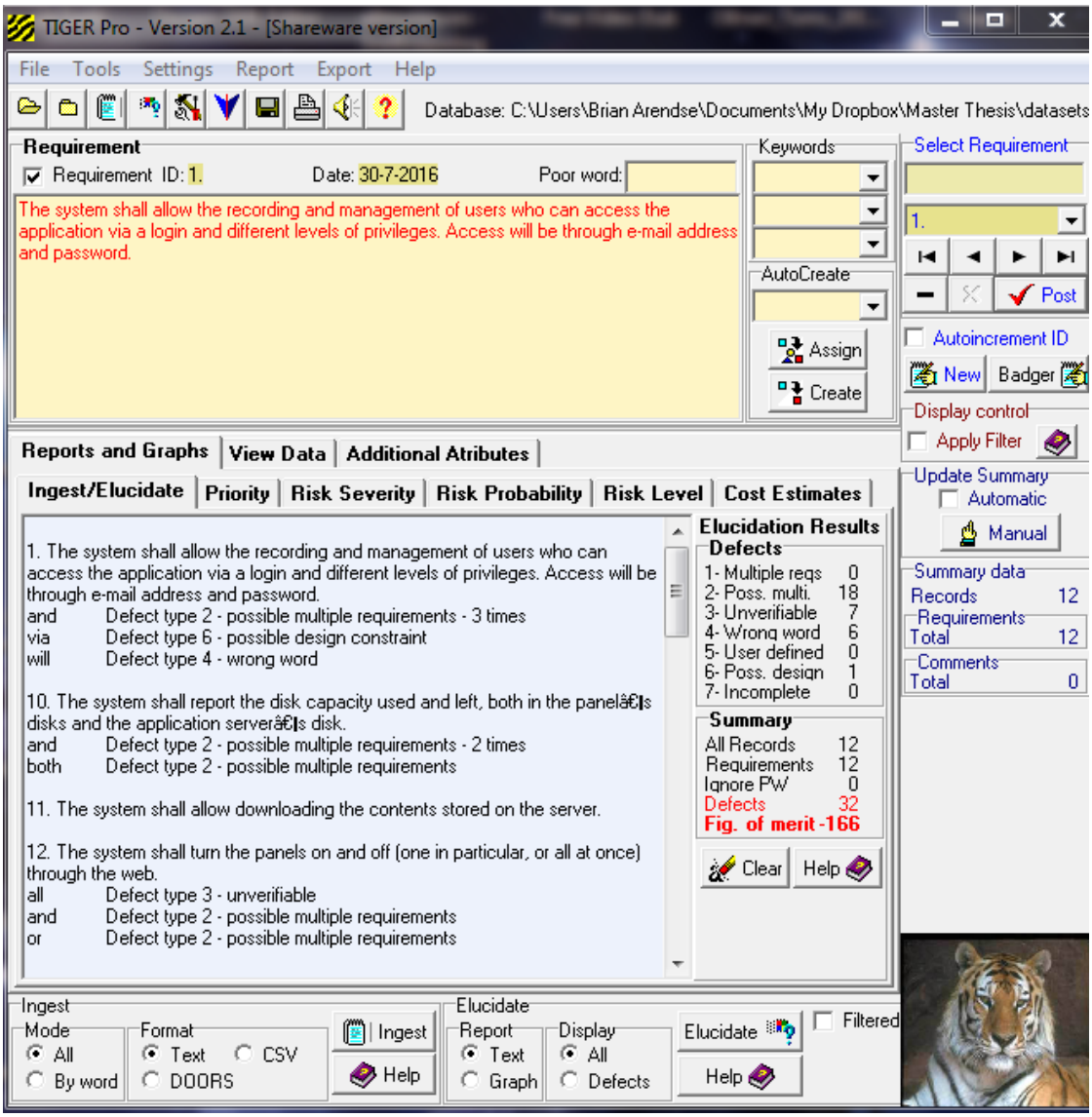

Figure 8 TIGER-PRO report

4.3 Metrics

The literature search for frameworks and metrics for measuring the performance of NLP tools did not yield much results. Looking at the evaluation methods of the identified NLP tools however provided some performance metrics that were frequently used; precision, recall, accuracy, and specificity. The benefit of these metrics is that it can be applied to a wide range of quality attributes such as ambiguity, duplicity, and consistency. To determine all of the performance metrics for a quality attribute the number of true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN) has to be determined. The individual performance metrics can be determined as followed (Nakache, Metais, & Timsit, 2005):

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

$$Specificity = \frac{TN}{TN + FP}$$

This chapter has provided an analysis of the 50 identified tools. This analysis shows that even on a parser level a lot of differences exist between NLP tools. Even that way in which corpus and dictionaries are used differs per tool. This chapter furthermore provided a more thorough analysis of the 3 selected tools. This chapter concludes with the explanation of the metrics available for the performance measurement of NLP tools.

5 Requirement standard

The purpose of the requirement standard is to have a standard to identify the defects in the datasets. A standard is important because this way the subjective involvement of the author (tagging the datasets) is brought to a minimum. Writing a requirement standard down also makes sure the research can be repeated and replicated.

The requirement standard consists of 17 features, 5 regarding atomicity, 11 regarding ambiguity, and 1 regarding passive and active voice. A feature is a functionality or defect check performed by at least 1 one the tools involved in the experiment. For each feature a definition is given, and in some cases accompanied by further explanation and examples. The description of each feature explains how and when a requirement defect can occur regarding that feature. The description of the features is based on how the 3 tools in the experiment define quality defects regarding those features. The handbook of ambiguity by Berry et al. (2003) is consulted in the case the tools contradict each other or the description of the tools is not clear enough. The upcoming section describes the requirement standard used in this research.

5.1 Requirement standard in this research

Atomicity

1. "and" – The occurrence of an *and* that describes multiple features.
2. "or" –The occurrence of an or that describes multiple features.
3. Multiple imperatives – The occurrence of more than one imperative describes multiple features.
   e.g. "shall" more than once.
4. List of items (enumeration) – The occurrence of a list of nouns or noun phrases that describes multiple features
   - Indicators could be
     - ": "
     - "-"
     - ","
     - Multiple "and"

   E.g. "The system must process text, images, and videos
5. Multiple sentences – The occurrence of multiple sentences that describes multiple features
   - Only 1 instance of this can occur in a requirement

Ambiguity

1. Vague pronouns – The occurrence of a pronoun of which it is not certain to which noun or noun phrase the pronoun refers to. (referential ambiguity)(Berry et al., 2003)
   - Proper pronouns are terms that substitutes a noun or a noun phrase, and do not refer to a concept or idea.

- A pronoun at the beginning of a sentence is a strong indicator that the pronoun refers to a concept or idea. ((Berry et al., 2003))

2. Quantifier – The occurrence of a quantifier of which the scope is not certain.
   - Universal quantifiers have a high likelihood to be ambiguous (all, both, any, each, every, some, and one).
     - "simply because very few universal statements about the world have no exceptions" (Berry & Kamsties, 2000)
   - The use of "a" or "an" could be an universal quantifier.
     E.g. "An office has a door connecting the office to a hallway" implies that there could be more than one office, door, and hallway. "An" in this case could be interpreted as a universal "each", so that each office has a door… ((Berry et al., 2003))
   - Checking for quantifier is closely related to Scope ambiguity (Berry et al., 2003):
     - "All linguistic prefer a theory", scope can be on "all" and on "a"
     - "No one has seen a pig with wings", scope can be on "no one" and on "a"
3. Unit – The occurrence of a number without a unit.
   - This includes both written and numeral numbers
4. Slash – The occurrence of a slash of which the nouns or noun phrases on either side have a different meaning.
   - This includes attached and separated slashes
     E.g. passengers/customers and passengers / customers
5. Non-verifiable terms – The occurrence of terms of which the truth or accuracy cannot be determined.
   - Terms that are hard to measure (e.g. unlimited, correct). But not if the non-verifiable term can be measured by the context.
   - The truth or accuracy cannot be demonstrated
   - These are terms that are used in conjunction with a noun or noun phrase (e.g. unlimited users, maximum response time, user friendly interface)
     E.g. "The maximum number of users is 1000 users". Can there be 1000 users or at most 999 users?
6. Subjective terms – The occurrence of terms which people can interpret differently.
   - Subjective terms are different from non-verifiable terms in a way that the latter is harder to quantify by subjective interpretation.
   - Some terms could be:
     - a few
     - appropriate
     - effective
     - efficient
     - sufficient
7. Comparative terms
   - Terms that rely on something outside of the requirements (more, less) or something mentioned earlier.

E.g. “The response time shall be more than it was before”. But also “The number of users shall be more than one” because it is not clear if a single user is also allowed. Replace “more” with “no less than x or greater than x” or “no greater than x or less than x”

- Elliptical ambiguity (Berry et al., 2003):
  - Perot knows a richer man than Trump

8. Negative terms
   - Terms imply a constraint rather than a requirement
   - Some terms could be:
     - cannot
     - Can't
     - no
     - not
9. Superlative terms (highest, lowest)
   - Terms that signify the greatest form of a descriptor
   - Relating to, or nothing the highest degree of the comparison of adjectives and adverbs”

     E.g. “The system shall have the highest response time”. But not “The system shall display the 10 airplanes with the earliest departure times”
10. Ambiguous words check (ontology)
    - If a term does not fit within one of the above mentioned metrics, but it is ambiguous, it is classified as “Ambiguity”. These are mostly adjectives and adverbs.
    - Pay extra attention to:
      - And (coordination ambiguity)
      - Any
      - Include
      - After, before, next and previous
      - Minimum
      - Maximum
      - Or (coordination ambiguity)
      - For up to (including or excluding)
    - Analytical ambiguity: Multiple adverbs or adjectives before a noun (Berry et al., 2003)
      - E.g. the Tibetan history teacher
    - Attachment ambiguity: a phrase that can modify both a verb and a noun in a sentence (Berry et al., 2003)
      - the police shot the rioters with guns
    - To be correct, an “only” should be immediately preceding the word or phrase that it limits (Berry et al., 2003).
    - “Also” suffers the same fate as only in that it is supposed to be put immediately preceding the word or phrase it modifies
    - There are other words that have the same problem as “only” and “also”. These include
      - almost

- even
- hardly
- just
- merely
- nearly
- really

11. Escape clause sentences
    - Sentences that have an open ending
        - as little as possible
        - as much as possible
        - if it should prove necessary
        - if practicable
        - so far as is possible
        - where possible
        - where there is sufficient space

Passive voice

- Verbs that imply that something has already happened. [2]

  E.g. “A user has been notified before something happened”. But not “A user is notified if something happens”.

- “To know whether you are writing in the active or passive voice, identify the subject of the sentence and decide whether the subject is doing the action or being acted upon.” (Femmer, Kuˇ, & Vetrò, 2014).
    - A sentence is passive voice if the subject of the sentence is the receiver of the action performed in the sentence.
    - A sentence is active voice if the subject of the sentence performs the action in the sentence.

[2] http://www.plainlanguage.gov/howto/quickreference/dash/dashactive.cfm

6 Experiment Conduction and Results

This chapter describes how the experiment is conducted and reports on the obtained results. The first section describes the conduction of the experiment. The second section shows the performance of the tools for each feature. The third section describes the approaches of the tools, followed by a comparison of the approaches.

6.1 Experiment Conduction

This section elaborates on the conduction of the experiment. Before the experiment starts a requirement standard is created to have a definition of how the quality of a requirement can be measured. Based on this requirement standard 4 datasets are manually tagged by the author, and validated by a requirements expert at the Utrecht University. The datasets are prepared for each tool so that the tools can process the datasets (e.g. excel files, txt files, and word files). The prepared dataset are given as input to the tool involved in the experiment. The output of the tools can be found in Arendse & Lucassen (2016). The difference between the manually tagged datasets and the output of the tools is used to calculate the performance of the tools. The performance of the tools is used to qualitatively compare the approaches of the tools.

6.2 Tool performance

For each feature the performance is measured by using a micro average of the 4 datasets. Micro average means that the true positives, false positives, true negatives, and false negatives for every dataset are summed up before the calculation of precision and recall (Tague-Sutcliffe, 1992). Another method is using the macro average. With this method the precision and recall is calculated for each dataset, after which the average of the precision and recall for all the datasets is averaged. In this thesis the micro average is used because of 3 reasons. First the number of datasets in this research is relatively small. When using the macro average this means that the performance of an individual dataset could have a big impact on the average. One of the datasets used in this research consists of user stories. The performance of this dataset could differ from the other datasets, which skews the results. The second reason is that datasets in this research vary in size. Together with the number of datasets, this could furthermore skew the results towards the performance of an individual dataset. The final reason is that this research focusses on the performance of the individual features. Macro average is useful for measuring the overall performance of a system. Micro average on the other hand is more suited for the measurement of sub-parts of a system.

The performance results of the 3 selected tools on the 4 datasets are available in Table 4. The rows with grey highlighting depict the total performance for the constructs atomicity and ambiguity for each of the tools. The total of the construct atomicity consists of the features “and”, “or”, multiple imperatives, list of items, and multiple sentences. The total of the construct ambiguity consists of the features vague pronouns, quantifier, unit, slash, non-verifiable, subjective, comparative, negative, superlative, general ambiguity, and escape clause. The *specific* score indicates the tool’s performance only counting the features it has implemented, while the *overall* score averages all features described

in the requirement standard. An example of the *specific* score is that the *specific* score of TIGER-PRO for atomicity consists of the total of the features "and", "or", and multiple imperatives. The overall score for atomicity includes the features list of items and multiple sentences, even though TIGER-PRO does not have those features implemented. The precision and recall cell pairs with green background represent the best performing approach according to the tool's recall for that feature. The focus is on recall and not on precision because precision only measures what the tools do detect, where recall also measures what the tools do not detect. Furthermore it is the goal of most NLP tools to achieve 100% recall of quality defects, sacrificing precision if necessary: *The Perfect Recall Condition* (Berry et al., 2012)

Table 4 Tool performance

Total Score	Qualicen		TIGER		RQA	
	Precision	Recall	Precision	Recall	Precision	Recall
"and"	-	-	0.66	1	0.66	1
"or"	-	-	0.22	1	0.22	1
Multiple imperatives	-	-	1	0.6	0.96	1
List of items	-	-	-	-	0.47	0.8
Multiple sentences	-	-	-	-	1	0.56
Atomicity specific	-	-	0.70	0.85	0.72	0.88
Atomicity overall	-	-	0.70	0.66	0.72	0.88
Vague pronouns	0.44	0.25	-	-	0.76	0.91
Quantifier	-	-	0.95	0.67	0.76	0.5
Unit	-	-	-	-	0.1	1
Slash	1	1	1	0.5	1	1
Non-verifiable	0.4	1	0.5	1	0	0
Subjective	1	0.33	1	0.08	1	0.17
Comparative	0	0	1	0.75	1	0.14
Negative	0.83	1	-	-	1	1
Superlative	0.75	1	-	-	-	-
General ambiguity	0.38	0.29	-	-	0.65	0.24
Escape clause	-	-	0	0	1	0.43
Ambiguity specific	0.55	0.36	0.93	0.5	0.65	0.55
Ambiguity overall	0.55	0.26	0.93	0.23	0.65	0.52
Passive voice	0.92	0.89	-	-	0.98	0.66

6.3 Comparison of the approaches

"And"

Both TIGER-PRO and RQA consider the occurrence of an *and* as being a sign of no atomicity. Qualicen does not focus on atomicity. The performance of TIGER-PRO and RQA is the same, since they both

detected all the instances of *and*. Some instances of "and" imply ambiguity rather than atomicity defects, which explains the precision.

"Or"

Both TIGER-PRO and RQA consider the occurrence of an *or* as being a sign of no atomicity. Qualicen does not focus on atomicity. The performance of TIGER-PRO and RQA is the same, since they both detected all the instances of *or*.

Multiple Imperatives

TIGER-PRO and RQA both check for multiple imperatives in a requirement. Qualicen does not focus on atomicity. TIGER-PRO only checks for the imperative *shall*, which results in a precision of 100% and a recall of 60%. RQA additionally checks for the imperatives *might, can, will, must, and may*. These additional imperatives results in a precision of 96% and a recall of 100%.

List of Items

RQA is the only tool that checks for a list of items. Since RQA is a commercial tool the actual approach is confidential. Therefore the approach can only be analysed by looking at the instances of list of items RQA detects. The approach derived from this is that RQA checks for the occurrence of multiple *and with*in a sentence. The instances RQA missed were lists that don't use an Oxford comma while listing items. For example the requirement "The rental transaction is created, printed and stored" is not detected by RQA. The requirement could be read that the creation process of a rental transaction consists of the actions printing and storing. Another interpretation is that there are 3 actions for the rental transaction (created, printed, and stored).

Multiple Sentences

RQA is the only tool that checks for multiple sentences. It does so by looking at the occurrence of a period that marks the end of a sentence. In some cases however 2 sentences were connected by a comma instead of a period. RQA doesn't check for commas, which results in a precision of 100% and a recall of 56%.

Vague pronouns

Qualicen and RQA both check for vague pronouns, where TIGER-PRO does not. Qualicen checks for vague pronouns by using POS tagging. Initially every pronoun marked by the POS tagger. Qualicen, together with their clients, formulated some exception rules that filters a part of the pronouns. These exception rules consist of certain types of grammar of a sentence where pronouns are not vague or ambiguous. Since Qualicen is a commercial tool the exception rules are confidential. The approach of Qualicen however missed every instance of the pronouns *it* and *this*. The reason for this is that Qualicen only checks for relative pronouns Therefore Qualicen did detect the pronouns *that, what, where, which, who,* and *whose*.. The result is a precision of 44% and a recall of 25%.

RQA uses a dictionary to check for vague pronouns. The dictionary consists of *him, his, he, it, this, that, their, our, your, mine, we, I, they, you, us, her, she,* and *them.* Based on the dictionary RQA checks for personal pronouns, possessive pronouns, demonstrative pronouns, and some relative pronouns. In the

dataset consisting of user stories the detection of the pronoun *I* is ignored. The result is a precision 76% and a recall of 91%.

Quantifier

TIGER-PRO and RQA both check for quantifiers, Qualicen does not check for quantifiers. Both tools use a dictionary to check for quantifiers. The dictionary of RQA consists of *all, both, and any*, with a results of a 76 % precision and 50% recall. TIGER-PRO additionally checks for the term *each,* which results in a 95% precision and 67% recall.

Unit

RQA is the only tool that checks for numbers without units. RQA detects both written and numeral numbers. The approach of RQA marks all numbers as being ambiguous, irrespective if it's followed by a unit or a noun. The result is a precision of 10% and a recall of 100%.

Slash

All three tools check for slashes. Qualicen and RQA detect both slashes that are separate (e.g. passengers / customers) and slashes that are attached (e.g. passengers/customers). This approach results in a precision of 100% and a recall of 100%. TIGER-PRO only detects the separate slashes, with a results of 100% precision and 50% recall.

Non-verifiable terms

All three tools check for non-verifiable terms by using a dictionary. The dictionary of RQA consists of the 13 terms *prompt, fast, routine, maximum, minimum, optimum, nominal, easy to use, close quickly, high speed, medium sized, best practices,* and *and user friendly*. The result is a precision and recall of 0% since RQA didn't detect any non-verifiable terms in the datasets.

The dictionary of TIGER-PRO consists of the 17 terms *affect, all, any, as little as, each, best practice, maximize, maximum, minimize, minimum, quick, rapid, user-friendly, sufficient, include, includes,* and *including*. The result is a precision of 50% and a recall of 100%.

The dictionary of Qualicen consists of 7 terms. Because Qualicen is a commercial tool the content of the dictionaries is not available. A way to find out what is in the dictionary is to provide the tool with an extensive list of non-verifiable terms. The non-verifiable terms that are marked are in the dictionary of Qualicen. The developers of Qualicen however asked to be careful with the proprietary content of Qualicen. Therefore only the number of terms in the dictionary are used. The result is a precision of 40% and a recall of 100%

Subjective terms

All three tools check for subjective terms by using a dictionary. The dictionary of Qualicen consists of 25 terms, which results in a precision of 100% and a recall of 33%. The dictionary of RQA consists of the 25 terms *probably, perhaps, typically, normally, often, generally, usually, commonly, optionally, maybe, may, can, frequently, almost always, rarely, at last,* and *almost*. The result is a precision of 100% and a recall of 17%.

The dictionary of TIGER-PRO consists of the 32 terms accurate reconstruction *adequate, appropriate, brief, clear, easy, effective, enhance, enough, flexibility, flexible enough, high, high-activity, high-dynamic, high-priority, improve, large, little, low, many, optimized, periodic, possible, several, short-duration, significant, statistically monitor, world-class, irregular, unexpected, unusual,* and *good*. The result is a precision of 100% and a recall of 8%.

Comparative terms

TIGER-PRO and Qualicen check for comparative terms. RQA has no measures or dictionary for the detection on comparative terms. RQA however did identify one instance of comparative terms. A comparative term was part of another dictionary. The result is a precision of 100% and a recall of 14%.

TIGER-PRO uses a dictionary consisting of the 3 terms *also, same,* and *such as*. With these terms TIGER-PRO has a precision of 100% and a recall of 75%. Qualicen uses POS tagging and morphological analysis for the detection of comparative terms. This approach however did not detect any instance of comparative terms. The result is a precision and recall of 0%.

Negative terms

Qualicen and RQA check for negative terms, TIGER-PRO does not. Both tools use a dictionary to detect negative terms. The dictionary of Qualicen consists of 13 terms, which results in a precision of 83% and a recall of 100%.

The dictionary of RQA consists of the 17 terms *nothing, nobody, none, never, doesn't, won't, shan't, mustn't, couldn't, shouldn't, oughtn't, can't, no, nor, non, not,* and *cannot*. The result is a precision and recall of 100%.

Superlative terms

RQA is the only tool that checks for superlative terms. The tool does so by performing morphological analysis to detect the greatest form of an adverb or adjective. This approach results in a precision of 75% and a recall of 100%.

General ambiguity

Qualicen and RQA check for general ambiguity, TIGER-PRO does not check for this. Both tools use a dictionary to check for general ambiguity. The dictionary of Qualicen consists of 358 terms. The result is a precision of 38% and a recall of 29%.

The dictionary of RQA consists of the 109 terms *bad, good, too, worst, better, timely, provide for, normal, as required, effective, easy to, capability to, capability of, be capable of, as appropriate, adequate, safe, reliable, improved, efficient, as possible, approximately, flexible, versatile, user friendly, if practical, easy, not limited to, as a maximum, as a minimum, quick, sufficient, minimize, maximize, user-friendly, a few, a lot of, about, all, ancillary, any, appropriate, approximate, as far as possible, as little as possible, as much as possible, as necessary, at least, based on, best possible, best practices, both, clearly, close quickly, close to, common, customary, easily, easy to use, enough, extremely, fault tolerant, few, first rate, generic, great, high fidelity, if it should prove necessary, large, little, manage,*

many, many of, maximum, medium-sized, minimal, minimum, necessary, optimal, optimize, optimum, proficient, prompt, quickly, reasonable, relevant, robust, roughly, routine, satisfactory, several, significant, small, so far as possible, some, state of the art, sufficiently, suitable, typical, typically, useable, vague, very nearly, rapid, high speed, slow, fast, friendly, and *more or less*. The result is a precision of 65% and a recall of 24%.

Escape clause

TIGER-PRO and RQA check for escape clauses, Qualicen does not do this. TIGER-PRO has no explicit dictionary for escape clause. There are however several escape clause terms in the dictionary of non-verifiable terms. These terms are *e.g, i.e., etc.,* and *eg*. The result is a precision and recall of 0%.

RQA also uses a dictionary for the detection of escape clauses. The dictionary consists of the 25 terms *among others, as a minimum, not limited to, not determined, not defined, tbc, tbs, tbd, further, etcetera, etc., and so on, shall be included but not limited to, but not limited to, e.g., eg, example, such as, if possible, if required, possibly, various, when requested by, when required,* and *to be determined*. The results is a precision of 100% and a recall of 43%

Passive voice

Qualicen and RQA check for passive voice, TIGER-PRO does not. The approach of both tools is protected by the tools. Analysing the results of the experiment does provide some insight in how the tools operate. The approach of Qualicen most likely uses POS tagging and morphological analysis. Adverbs and adjectives that might indicate passive voice (e.g. the requested book) were ignored by Qualicen, where verbs were detected (e.g. shall be requested). The results is a precision of 92% and a precision of 89%.

RQA detects every instance of a verb, adverbs or adjective that is written in past tense. The morphological analysis is probably different than that of Qualicen. RQA failed to detect the instance *can be read*, where Qualicen did detect this. It could be that Qualicen uses the word *be* as in indicator, which is often true. RQA does not do this and has less true positives and false positives. Therefore the approach of RQA results in a precision of 98% and a recall of 66%

6.4 Good and bad practices

This section describes a set of good and bad practices derived from the results. The good and bad practices can be used by NLP tool developers for the creation and improvement of NLP tools. Please note that these are based on the results of an experiment including 3 tools. Developers have to consider whether the good and bad practices apply to their tools. The 3 sections in this chapter each describe a good or bad practice.

Different tokenizers

The choice of which tokenizer to use can have an effect on the performance of a tool. Especially when it comes to symbols the approaches of tokenizers differ. The end of a sentence is in most cases marked with a period. Abbreviations can also be closed with a period. Some tokenizers use a corpus to detect

these abbreviations. Other tokenizers use the structure of a sentence to determine whether a period marks the end of a sentence or the end of a word. Finally there are tokenizers that consider each period as the end of a sentence. For the detection of the feature *Multiple sentences* the choice of tokenizer is extremely important. Using a tokenizer that considers each period the end of a sentence is likely to results in a low precision but high recall. Using one of the other approaches is more likely to result in a better precision, but it could have a negative effect on the recall.

Another feature effected by the choice of tokenizer is *Slash detection.* RQA and Qualicen both use a tokenizer that detects slashes in between or at the end of words (e.g. passengers/customers). TIGER-PRO however only detects slashes that are separated from words (e.g. passengers / customers). The result is that all tools have a precision of 100%. RQA and Qualicen have a recall of 100%, where TIGER-PRO's recall is 50%.

Dictionary vs. Parsing

Most features regarding ambiguity use a dictionary to detect any ambiguous terms. There are however several features, such as *vague pronouns* and *comparative terms*, that use parsing to detect ambiguity. The strength of using a dictionary is that a comprehensive list can be created to detect any defects in requirements. The dictionary can even be adjusted to a certain domain to improve the performance. Parsing on the other hand is more delicate approach. Possible defect are marked by the parser after which exception rules filters out instances that use a certain types of grammar.

For the feature *vague pronouns* RQA uses a dictionary. Qualicen uses a POS tagger to detect vague pronouns. The performance of RQA is better in this experiment, for both precision and recall. The approach of Qualicen has a relatively low precision and recall, and missed all instance of the most common pronouns *it* and *this*. This could be due to two reasons. First the POS tagger of Qualicen could not be working properly, and pronouns were not tagged as being pronouns. Using a different parser could help resolve this problem. Second the exception rules could be too strict by which true defects were wrongfully filtered out. These exception rules however were created together with clients of Qualicen. It is possible that only instances of vague pronouns are detected that they consider to be ambiguous. This more practical approach however does not perform well in this experiment.

For the feature *comparative terms* TIGER-PRO uses a dictionary. Qualicen uses a POS tagger and morphological analysis to detect comparative terms. Even though the dictionary of TIGER-PRO only consists of 3 terms, it managed to get a 75% recall. It has to be noted that the number of comparative terms in the datasets is limited to only 7 instances. Nevertheless the approach of Qualicen is not able to detect any instances of comparative terms. This could be because of the limited instances of comparative terms. Another reason could be that the POS tagging or morphological analysis is not performed properly. Because the approach of Qualicen is proprietary it cannot be said with certainty what the real reason is that the approach does not perform well. All that can be said is that the dictionary approach of TIGER-PRO performs better than the parsing approach of Qualicen.

What is in the dictionary

All three tools utilize dictionaries for multiple features. For both RQA and TIGER-PRO this is the main approach for detecting ambiguity and atomicity defects. The difference in performance of the dictionaries can be explained by what is in the dictionaries and the size of the dictionaries. Even with 1 term in the multiple imperatives dictionary TIGER-PRO has a recall of 60%. RQA uses 6 terms however and has a recall of 100%. The same goes for quantifiers. RQA checks for *all, both,* and *any*, and TIGER-PRO checks additionally for *each.* Using only this one extra term results in a both higher precision and recall (0.19 and 0.17). Finally for the detection of negative terms Qualicen uses 13 terms and RQA uses 17 terms. Both tools have a 100% recall, RQA however has a higher precision (0.17) even with more terms in the dictionary. This indicates that the bigger the dictionary does not necessarily mean a lower precision. It furthermore shows that what is in the dictionary effects the precision and recall of a tool.

It has to be noted that for some features it is easier to create an exhaustive list of all the terms than other features. There is for instance a limited number of pronouns, imperatives, and quantifiers. RQA checks for all imperatives and therefore has a 100% recall regarding multiple imperatives. RQA also does this for pronouns. They do not check however for all pronouns such as *what, where, which, who,* and *whose*. Nevertheless the tool has a precision of 91%. For the quantifiers it is more complicated. TIGER-PRO and RQA both check for universal quantifiers. With that approach they manage to detect more than half of the quantifier defects. In some cases however the terms *a, an,* and *the* are used as quantifiers that cause ambiguity. If the tools were to check for these terms the recall would be higher, and possibly even 100%. The precision however would be severely effected in a negative way. A more comprehensive approach is desired, where instance are filtered out using exclusion rules. None of the tools in the experiment however has found an approach for this.

The number of subjective and non-verifiable terms is a harder to define. One reason for this is that the definition for what a subjective term or non-verifiable term is differs per tool. TIGER-PRO has a dictionary for non-verifiable terms, while it actually consists of subjective terms, non-verifiable terms, comparative terms, escape clauses, and quantifiers. Some terms can furthermore be classified as more than one type of term. The main property of non-verifiable terms is that they are hard to measure. Because of this people can use their own measurement for these kind of terms, making it also a subjective term.

7 Design of the next generation tool

This chapter describes the design of the next generation tool. The first section elaborates on the best performing approaches of the tools involved in the experiment. The second section describes how these approaches can be implemented into a tool.

The results of the experiment and their analysis in Chapter 7 serve as a design blueprint for the next generation tool. The results of the experiment, together with the technical information about the tools provided by their developers, lead to the design of the best approaches for each feature. These approaches can be implemented into new or existing NLP tools to increase the performance of these tools. Please note that the results are based on an experiment involving 3 tools. Developers have to consider and test themselves whether the approaches described in this chapter actually outperform the current approaches of their NLP tools.

7.1 Approaches

This section describes the best performing approaches of the 3 tools involved in the experiment. The approaches of the next generation tool are:

1. “and” – Any instance of *and*

2. “or” – Any instance of *or*

3. Multiple imperatives – A dictionary with the imperatives *shall, might, can, will, must, and may*

4. List of items – Multiple *and* in a sentence.

5. Multiple sentences – Multiple “.” that indicate the end of sentence

6. Vague pronouns – A dictionary with the pronouns *him, his, he, it, this, that, their, our, your, my, we, I, they, you, us, her, she, and them*

7. Quantifier – A dictionary with the quantifiers *all, both, any*, and *each*

8. Unit – Any instance of a written or natural number

9. Slash – A tokenizer that detects the symbol / without space between the words on either side

10. Non-verifiable terms – A dictionary with the terms *affect, all, any, as little as, each, best practice, maximize, maximum, minimize, minimum, quick, rapid, user-friendly, sufficient, include, includes, and including*

11. Subjective – The dictionary of Qualicen with 25 terms

12. Comparative terms – A dictionary with the comparative terms *also, same*, and *such as*

13. Negative terms – A dictionary with the negative terms *nothing, nobody, none, never, doesn't, won't, shan't, mustn't, couldn't, shouldn't, oughtn't, can't, no, nor, non, not, and cannot*

14. Superlative terms – Morphological analysis that detect the greatest form of adverbs and adjectives

15. General ambiguity – The dictionary of Qualicen with 358 terms

16. Escape clause – A dictionary with the terms *among others, as a minimum, not limited to, not determined, not defined, tbc, tbs, tbd, further, etcetera, etc, and so on, shall be included but not limited to, but not limited to, e.g., eg, example, such as, if possible, if required, possibly, various, when requested by, when required, and to be determined*

17. Passive voice – The parsing approach of Qualicen

In theory, a next generation tool that implements these 16 approaches should obtain 0.92 recall and 0.68 precision for atomicity, 0.76 recall and 0.77 precision for ambiguity, and 0.89 recall and 0.92 precision for passive voice. This is calculated by taking the micro average of the performance of the approaches included in the design of the next generation tool. However, far superior approaches might already be available that cannot be included because the design is based on already available tools. After all, the design is based on tools that are between 13 and 2 years old. For example, it is not inconceivable that new tools employing POS tagging for the detection of vague pronouns outperform the dictionary-based approach selected for the vague pronoun feature. Perhaps a combination of the two approaches is preferable. Furthermore, considering that POS tagging technology is continuously improving it is impossible to predict the exact performance of a next generation tool. Because of this, next generation tools should undergo a performance test against the same datasets used in this experiment (Arendse & Lucassen, 2016) .

7.2 Tool mashup

One should proceed with caution when starting the development of this next generation tool design. Unfortunately, it is impossible to combine the best-performing features into one tool without recreating the entire technology. In fact, any new custom-built tool released after years of development will likely be yet another tool that has its own strengths and weaknesses in terms of features. As research community, we should start a shared initiative to create a common, modular standard that allows for easily replacing old approaches with superior techniques. The right architecture design could even empower automatic tool assembly with the best modules for a specific context.

An example of a mash-up initiative is the UIMA Framework (Ferrucci & Lally, 2004), which enables developers to create unstructured information analysis tools by combining application components. The framework defines a standard interface that manages data flow between all available components. Developers can choose to configure existing annotators and repositories, or create new ones for their application. Through component re-use, developers can more quickly develop tools or

web applications that process unstructured text, audio and video. A simplified architecture of this framework is shown in Figure 9.

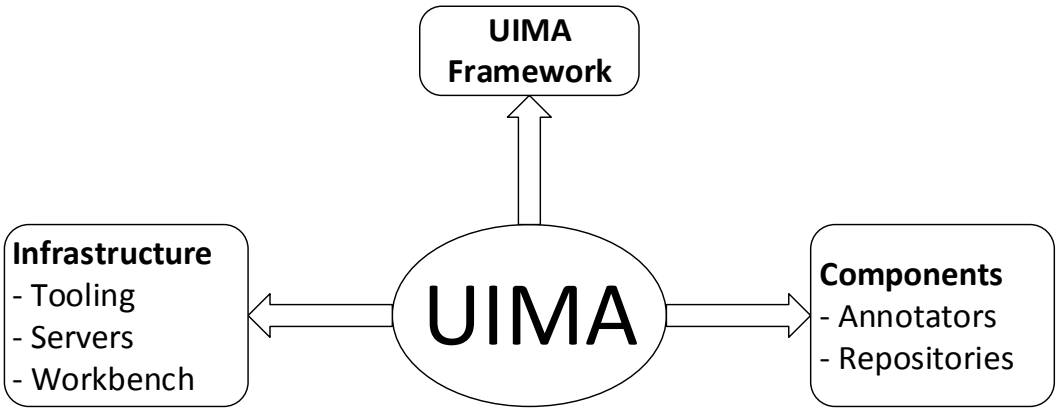

Figure 9 UIMA framework

Unfortunately, the UIMA Framework requires strong technical capabilities of the user to run it, despite the availability of user guides and tutorials. Achieving widespread adoption requires a user-friendly application with little set-up time or configuration. Ideally, the application exposes a cloud-based interface to allow a wider range of application integrations. It could even built upon existing mashup services such as Zapier[3] that enable non-technical users to easily integrate, automate and innovate by moving data between applications.

A mashup framework for the integration of NLP approaches into a tool would have to be easily accessible for less experienced developers. It becomes easier for researchers and the business environment to contribute to the body of knowledge regarding NLP in RE. Preferably the framework is so easy to use that all a user has to do is select the approaches he wants to implement into his tool. For this a unified user interface for reporting defects is required. The framework however should be flexible enough so that it supports various technologies and programming languages. Users have to be able to implement their approaches as intended, and not with the limitations of an enforced programming language.

[3] https://zapier.com/

8 Discussion

This chapter concludes this thesis by answering the research questions, presents limitations and issues for future work.

8.1 Conclusions

This thesis was set out to get a clear overview of the performance of the main approaches taken by NLP tools in the RE landscape, and to create a theoretical tool that synergistically integrates the best approaches. A literature study identified the 50 main NLP tools in the RE landscape. Based on exclusion criteria 3 tools remained to be included in the experiment. A requirement standard is created to manually process 4 datasets provided by the developers of the tools. Using these 4 manually tagged datasets the performance of the approaches for each feature of the tools is measured. From the results a theoretical next generation tool is designed based on the best performing approach for each feature. A set of good and bad practises is derived from the results of the experiment and the theoretical next generation tool.

The main research question of this thesis is "*How to create a best of breed Requirements Engineering Tool for improving the quality of requirements through Natural Language Processing?".* To answer this main research question there are 4 sub questions. The first sub question seeks for the similarities and differences between NLP tools. The 50 identified tools are analysed in chapter 4 to show the similarities and differences. The main difference of the tools is the technologies the tools use, and how they are used. The second sub question was set to identify the metrics for measuring the performance of NLP tools. A set of 4 metrics is presented in section 4.3 consisting of precision, recall, accuracy, and specificity is identified as being the most frequently used approach for measuring the performance of NLP tools. The third research question seeks out what approaches NLP tools can take to improve the quality of requirements. This question is answered in chapter 6 where the approaches of the 3 tools involved in the experiment are thoroughly analysed and compared. The final sub question provides a set of good and bad practices that other researchers can use for the development of their own NLP tool. This question is answered in section 6.4 where the following good and bad practices are described:

1. Different tokenizers: The choice to which tokenizer to use can have an effect (both positive and negative) on the performance of a tool
2. Dictionary vs. Parsing: Using a dictionary is a safe and simple method to detect defects. Parsing is a more complicated approaches, and when not performed correctly it can have a negative effect on the performance of a tool
3. What is in the dictionary: The size and content of a dictionary can have an effect on the performance (both recall and precision) of a tool, the bigger the dictionary, the better

The process of answering the sub questions provides the answer to the main research question. The result is the theoretical design of the next generation tool. NLP tool developers can incorporate these approaches into their own tool. For existing tools developers have to determine whether these

approaches fit within the framework of their tool. Furthermore they have to test whether these approaches perform better than their current approaches.

A final contribution made by this thesis is the requirement standard. Other researchers can use the requirement standard from this thesis to manually tag their own datasets. Even if their tools only cover a part of the requirement standard they can use it in the process of measuring the performance of NLP tools.

8.2 Limitations

After the research is conducted it is important to see how valid the results of the research are. For this there are 2 things to consider: reliability and validity (Golafshani, 2003). Reliability refers to the extent the results of the research remains the same over a period of time, and whether the research can be reproduced using the same method. A threat to the reliability for this research is the creation of the requirement standard. The standard is created by the author based on the definitions given by the tools and the handbook of ambiguity (Berry et al., 2003). If this research were to be replicated with other tools the requirement standard could take a different form than the one in this research, which leads to datasets that are tagged in another way. This brings forward a second threat to the reliability of this research; the tagging of the datasets. The datasets were manually tagged by the author. If another researcher were to tag the same datasets, with the same requirement standard, this could still lead to differently tagged datasets.

Validity refers to whether the research actually measures or achieves what was intended. In this thesis there are 3 main threats to validity: the definition of ambiguity and atomicity, the number of tools involved in the experiment, and the number of datasets used.

The first threat to validity is how the constructs of ambiguity and atomicity are created. They are created on how the 50 identified tools, and especially the 3 tools involved in the experiment, define ambiguity and atomicity. The author deducted the constructs ambiguity and atomicity from these definitions. It is not validated however that these construct actually measure the same thing, i.e. ambiguity and atomicity. The involvement and judgement of a researcher, in this case the author, is a threat to the construct validity of this research.

The second threat to validity is the number of tools involved in the experiment is limited to only 3 available tools, out of the 12 that where eligible. Involving more tools in the experiment could lead to a different design of the next generation tool. Therefore it is difficult to generalize the results, which is a threat to the external validity of this thesis. The number of approaches that are compared is limited, which in turn has an effect on the design of the next generation tool. Including more tools means more approaches, which could lead to different approaches in the design of the next generation tool.

A third threat to the validity of this research is the number of datasets involved in this thesis. Manually tagging datasets is a time consuming task. Therefore only 4 datasets are used from the tools involved in the experiment. This is threat to the internal validity of this thesis. Using more datasets could provide

the tools in the experiment with more terms (e.g. subjective terms). In turn this could lead to a different performance of the tools, and approaches, and a different design of the next generation tool.

8.3 Future work

A first issue that requires further work relates to the number of tools that are available to this thesis. The limitation that only 3 tools are available is at the same time an issue that should be solved by the research community. Making sure that a NLP tool can be used years after development increases both the reproducibility of that research and the progress that can be made in the field of NLP in RE. Performing the same research as in this thesis but only with more tools could significantly increase the validity of the results. The research community can solve this problem by using a standardised framework or platform, such as UIMA, to more easily create and mashup NLP tools.

The creation of a tool mashup approach is a second issue for future work. An accessible framework is required even though a mashup framework already exists in the form of UIMA. The focus of such a mashup framework should be on both easily integrating existing approaches into a tool as well as implementing and testing newly developed approaches. More progress in the field of NLP in RE could be made by making the framework accessible for less experienced developers.

A third issue for future work relates to the tagging of the datasets. This is a time consuming task and performed for every performance testing of NLP tools. Creating a database with tagged datasets that are validated could fasten the process of NLP tools testing. Such an initiative has already been started in the form of the NLRP Benchmark[4]. The problem with this initiative is that the number of (tagged) datasets is limited. Contributions made by the research community could make sure that the NLRP Benchmark becomes a valuable resource for every NLP tool developer.

A fourth issue for future work is researching the progress made in the semantic and pragmatic curve. This thesis focussed on syntactic NLP approaches in RE. Even though more research is needed to fully analyse the syntactic curve, it is already necessary to look at the semantic and pragmatic curve. The method used in this thesis however is probably not suited for studies of the semantic and pragmatic curve. Especially the definition of what a quality defect is and what would be semantically or pragmatically correct needs to be clearly defined.

A final issue for future work is the implementation of the approaches of the design of the next generation tool. In this thesis a mostly quantitative method is used to create the design of the next generation tool. But how will the users of real world RE systems will experience those approaches? Case studies are required to find out whether the approaches presented in this thesis are perceived during actual development projects.

[4] http://nlrp.ipd.kit.edu/index.php/Main_Page

9 References

Ambriola, V. (2006). On the Systematic Analysis of Natural Language Requirements with CIRCE. *Automated Software Engineering*, *13*(1), 107–167. doi:10.1007/s10515-006-5468-2

Ambriola, V., & Gervasi, V. (1997). Processing natural language requirements. *Proceedings 12th IEEE International Conference Automated Software Engineering*, 36–45. doi:10.1109/ASE.1997.632822

Arendse, B., & Lucassen, G. (2016). NLP RE Tools data repository. Retrieved from http://www.staff.science.uu.nl/~lucas001/nlp_tools/

Berezin, T. (1999). Writing a Software Requirements Document.

Berry, D., Gacitua, R., Sawyer, P., & Tjong, S. F. (2012). The case for dumb requirements engineering tools. *Requirements Engineering: Foundation for Software Quality*, 211–217. doi:10.1007/978-3-642-28714-5_18

Berry, D. M., & Kamsties, E. (2000). "The Dangerous 'All' in Specifications." *Proceedings of 10th International Workshop on Software Specification & Design*, 191–194.

Berry, D. M., Kamsties, E., Krieger, M. M., Loh, W., Lee, S., Angeles, L., … Krieger, M. M. (2003). From Contract Drafting to Software Specification : Linguistic Sources of Ambiguity. *Technical Document*, 1–80.

Brun, E., Steinar Saetre, A., & Gjelsvik, M. (2009). Classification of ambiguity in new product development projects. *European Journal of Innovation Management*, *12*(1), 62–85. doi:10.1108/14601060910928175

Cambria, E., & White, B. (2014). Jumping NLP Curves: A Review of Natural Language Processing Research [Review Article]. *IEEE Computational Intelligence Magazine*, *9*(2), 48–57. doi:10.1109/MCI.2014.2307227

Chantree, F. J., Roeck, A. de, Nuseibeh, B., & Willis, A. (2006). Identifying Nocuous Ambiguity in Natural Language Requirements. *Doctoral Dissertation*. doi:10.1109/RE.2006.31

Chomsky, N. (1956). Three models for the description of language. *IRE Transactions on Information Theory*, *2*(3), 113–124. doi:10.1109/TIT.1956.1056813

Dale, R., Moisl, H., & Somers, H. (2000). *Handbook of natural language processing*.

Davis, A. M., & Zowghi, D. (2006). Good requirements practices are neither necessary nor sufficient. *Requirements Engineering*, *11*(1), 1–3. doi:10.1007/s00766-004-0206-4

de Bruijn, F., & Dekkers, H. L. (2010). Ambiguity in natural language software requirements: A case study. *Requirements Engineering: Foundation for Software Quality*, 233–247. Retrieved from

http://upcommons.upc.edu/pfc/bitstream/2099.1/6255/1/memoria.pdf

Dingsøyr, T., Nerur, S., Balijepally, V., & Moe, N. B. (2012). A decade of agile methodologies: Towards explaining agile software development. *Journal of Systems and Software*, *85*(6), 1213–1221. doi:10.1016/j.jss.2012.02.033

Femmer, H., Fernández, D. M., Juergens, E., Klose, M., Zimmer, I., & Zimmer, J. (2014). Rapid Requirements Checks with Requirements Smells: Two Case Studies. *Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering*, 10–19. doi:10.1145/2593812.2593817

Femmer, H., Kuˇ, J., & Vetrò, A. (2014). On The Impact of Passive Voice Requirements on Domain Modelling, 1–4.

Ferilli, S., & Singh, S. (2011). *Automatic Digital Document Processing and Management*. doi:10.1007/978-3-642-15992-3

Ferrucci, D., & Lally, A. (2004). UIMA : An Architectural Approach to Unstructured Information Processing in the Corporate Research Environment. *Natural Language Engineering*, *10*(3), 327–348.

Firesmith, D. (2003). Specifying Good Requirements. *Journal of Object Technology*, *2*(4), 77–87. doi:10.5381/jot.2003.2.4.c7

Génova, G., Carlos, U., Madrid, I. I. I. De, Génova, G., Fuentes, J. M., Llorens, J., … Moreno, V. (2011). A Framework to Measure and Improve the Quality of Textual Requirements A Framework to Measure and Improve the Quality of Textual Requirements. *Requirements Engineering*, *18*(1), 25–41. doi:10.1007/s00766-011-0134-z

Gill, K. D., Raza, A., Zaidi, A. M., & Kiani, M. M. (2014). Semi-automation for ambiguity resolution in Open Source Software requirements. *Electrical and Computer Engineering (CCECE), 2014 IEEE 27th Canadian Conference on*, 1–6. doi:10.1109/CCECE.2014.6900955

Golafshani, N. (2003). Understanding Reliability and Validity in Qualitative Research, *8*(4), 597–606.

Harwell, R., Aslaksen, E., Hooks, I., Mengot, R., & Ptack, K. (1993). What is a Requirement? *INCOSE International Symposium*, *3*(1), 17–24.

Heflin, J. (2004). OWL Web Ontology Language Use Cases and Requirements. *W3C Recommendation*, *10*(12).

Ibrahim, M., & Ahmad, R. (2010). Class diagram extraction from textual requirements using natural language processing (NLP) techniques. *2nd International Conference on Computer Research and Development, ICCRD 2010*, 200–204. doi:10.1109/ICCRD.2010.71

Jain, P., Verma, K., Kass, A., & Vasquez, R. G. (2009). Automated Review of Natural Language

Requirements Documents: Generating Useful Warnings with User-extensible Glossaries Driving a Simple State Machine. *Proceedings of the {2Nd} India Software Engineering Conference*, 37–46. doi:10.1145/1506216.1506224

Kamsties, E., & Peach, B. (2000). Taming ambiguity in natural language requirements. *Proceedings of the 13th International Conference on System and Software Engineering and Their Applications*, *2*, 1–8. Retrieved from http://www.prof.kamsties.com/download/icssea2000.pdf

Kasser, J. (2003). Prototype Educational Tools for Systems and Software (PETS) Engineering, (October), 1–11.

Kiyavitskaya, N., Zeni, N., Mich, L., & Berry, D. M. (2008). Requirements for tools for ambiguity identification and measurement in natural language requirements specifications. *Requirements Engineering*, *13*(3), 207–239. doi:10.1007/s00766-008-0063-7

Kof, L. (2005). Natural Language Processing : Mature Enough for Requirements Documents Analysis ? *Natural Language Processing and Information Systems*, 91–102. doi:10.1007/11428817_9

Liddy, E. D. (2001). Natural Language Processing.

Lucassen, G., Dalpiaz, F., Brinkkemper, S., & van der Werf, J. M. E. M. (2015). Forging High-Quality User Stories: Towards a Discipline for Agile Requirements. *Proceedings of the IEEE International Requirements Engineering Conference*.

MacDonell, S., Min, K., & Connor, A. (2005). Autonomous requirements specification processing using. Retrieved from http://www.researchgate.net/profile/Andy_Connor/publication/264197862_Autonomous_requirements_specification_processing_using_natural_language_processing/links/54daf4e20cf261ce15cea33c.pdf

McGuinness, D. L., & van Harmelen, F. (2004). OWL Web Ontology Language Overview. *W3C Recommendation*, *10*(10). doi:10.1145/1295289.1295290

Mich, L., Franch, M., & Novi Inverardi, P. L. (2004). Market research for requirements analysis using linguistic tools. *Requirements Engineering*, *9*(2), 151–151. doi:10.1007/s00766-004-0195-3

Nadkarni, P. M., Ohno-Machado, L., & Chapman, W. W. (2011). Natural language processing: an introduction. *Journal of the American Medical Informatics Association*, *18*(5), 544–551. doi:10.1136/amiajnl-2011-000464

Nakache, D., Metais, E., & Timsit, J. F. (2005). Evaluation and NLP. *Database and Expert Systems Applications*, 626–632.

Paetsch, F., Eberlein, A., & Maurer, F. (2003). Requirements engineering and agile software development. *WET ICE 2003. Proceedings. Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003.*, 308–313.

doi:10.1109/ENABL.2003.1231428

Ryan, K. (1993). The role of natural language in requirements engineering. *Proceedings of the IEEE International Symposium on Requirements Engineering*, 240–242. doi:10.1109/ISRE.1993.324852

Tague-Sutcliffe, J. (1992). The pragmatics of information retrieval experimentation. *Information Processing & Management*, *28*(4), 467–490.

Voutilainen, A. (2003). *The Oxford handbook of computational linguistics*.

Weikum, G. (2002). Foundations of statistical natural language processing. *ACM SIGMOD Record*, *31*(3), 37. doi:10.1145/601858.601867

Wiegers, K. E. (1999). Writing Quality Requirements. *Software Development*, *7*(5), 44–48.

Xiao, X., Paradkar, A., Thummalapenta, S., & Xie, T. (2012). Automated extraction of security policies from natural-language software documents. *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering - FSE '12*, 1. doi:10.1145/2393596.2393608

Yang, H., Roeck, A. De, Gervasi, V., Willis, A., & Nuseibeh, B. (2011). Analysing Anaphoric Ambiguity in Natural Language Requirements. *Requirements Engineering*, *16*(3), 163–189.

Yang, H., Roeck, A. De, Gervasi, V., Willis, A., & Nuseibeh, B. (2012). Speculative requirements: Automatic detection of uncertainty in natural language requirements. *20th IEEE International Requirements Engineering Conference (RE), 2012*, 11–20.

Yue, T., Briand, L. C., & Labiche, Y. (2011). A systematic review of transformation approaches between user requirements and analysis models. *Requirements Engineering*, *16*(2), 75–99. doi:10.1007/s00766-010-0111-y

10 Appendix

10.1 Appendix A: Paper

The paper will be published in September 2016 in the workshop on Artificial Intelligence for Requirements Engineering (AIRE). The properly formatted paper can be found there.

Toward Tool Mashups: Comparing and Combining NLP RE Tools

Brian Arendse and Garm Lucassen

Utrecht University

Utrecht, The Netherlands

b.arendse@students.uu.nl, g.lucassen@uu.nl

Abstract—Over 50 papers present natural language pro-cessing tools for improving the quality of requirements. How-ever, few of these are adopted by industry. Even worse, most of them are no longer publicly available or supported by their creators. The few available and actively maintained tools exhibit some outstanding features, but also include sub-optimal functionalities. In this paper, we compare the performance of 3 existing tools on how well they automatically detect ambiguity and atomicity defects and deviations in 4 real-world natural language requirements sets. Next, we show how to design a superior tool by combining the best performing approaches of these three. Finally, we introduce a research roadmap toward automatically generating NLP RE tool mashups through the assembly of modular components taken from existing tools.

Index Terms—Requirements engineering, Natural language processing, Quality of requirements, Mashups

I. INTRODUCTION

The fundamental purpose of a software requirement is to convey a mental representation of an intended functionality or quality of a system to the reader of the requirement [11]. How-ever, human requirements engineers frequently make mistakes in the way they formulate requirements, causing expensive re-work in later phases of software engineering [4].

With the aim of mitigating these errors, the Requirements Engineering (RE) community has churned out 50+ tools in the past 15 years that leverage natural language processing (NLP) for (semi-)automatically improving requirements quality [1].

However, the specific facets of requirements quality these tools strive to improve varies substantially. The number and diversity of lists defining quality criteria and/or characteristics of software requirements further complicates matters [11, 8, 29]. Just to give a few examples, the ISO 29148 standard defines 13 characteristics of high quality system requirements specification [25], Pohl defines quality in terms of specification completeness, agreement among stakeholders, and formality of the representation [22], while Lucassen et al. [17] list 13 linguistic criteria that define high-quality user stories.

All NLP RE tools strive to improve requirement quality by focusing on one or more quality attributes from one or more criteria standards. Berry et al. [3] categorizes the fundamental approaches of all NLP RE tools into four types:

1. Finding defects and deviations in natural language (NL) requirements document;

2. Generating models from NL descriptions;

3. Inferring trace links between NL descriptions;

4. Identifying the key abstractions from NL documents.

In this paper, we compare the performance of 3 NLP RE tools of the first type, which find defects and deviations in NL requirements documents. We do so by experimentally testing their performance on four real-world requirements data sets. By analysing the results, we design a hybrid tool that combines the most effective approaches of each for state-of-the-art results.

For the purposes of this paper, we focus on two specific requirements quality characteristics: ambiguity and atomicity. Academic literature describes multiple approaches on how to automatically ensure the quality of these two requirements characteristics. Because reducing ambiguity is arguably a key objective of requirements engineering [11], much effort has been put into defining requirements ambiguity and developing tools that detect deviations. Furthermore, atomicity is clearly detectable and definable for both non-expert humans as well as machines.

On the other hand, other quality criteria—such as complete-ness, consistency or traceability—require deep understanding of the domain or access to metadata unspecified in the requirements text itself, thereby making them less suitable to our research, which relies on objective metrics of performance.

The rest of the paper is structured as follows. Section II further details the chosen quality aspects for requirements: ambiguity and atomicity. This is followed by an exposition of our experimental method in Section III and its results in Section IV. Section V presents our design for a next generation tool, followed by related work in Section V. We discuss the limitations of this study in Section VI and present our conclusions and directions for future work in Section VII.

II. AMBIGUITY AND ATOMICITY

Although minimizing ambiguity is a primary objective of requirements engineering, it is impossible to avoid ambiguity completely [2]. A requirement is ambiguous if it can have multiple interpretations [29, 5, 6, 10, 13]. Berry et al. [2] argue that 100% unambiguous requirements do not exist. There will always be someone with a different interpretation, especially in projects where there are multiple types of stakeholders involved such as business analysts, developers, high level management, and customers. This implies that a perfect requirement—more precisely, a perfectly formulated requirement—is an oxymoron: it does not exist. However, not all requirements are equally imperfect. Some requirements are understood in a similar way by the vast majority of stakeholders. One way to measure the

ambiguity of a requirement is the percentage of requirements engineers that form the same mental representation by reading the requirement. If every requirement engineer in the group has the same mental representation the requirement is unambiguous. Yet, any new requirements engineer might still form a different mental representation [2]. Furthermore, determining ambiguity in this way requires a lot of time investment and produces unpredictable results depending on the experience, capabilities, and background of the requirement engineers.

Instead, NLP RE Tools employ on a more quantitative approach by looking at the causes of ambiguity. Ambiguity can occur because of the use of words that are prone to be ambiguous. For instance, the interpretation of superlative terms (e.g. best and highest) and universal quantifiers (e.g., all, each, and every) is subjected to the view and state of mind of an individual. The number of ambiguous words can then quantify the ambiguity of (a set of) requirements.

A requirement is atomic when it specifies exactly one function. To find atomicity (often referred to as cohesiveness in the literature) violations, NLP RE tools detect requirements that specify more than one function of a system using a similar approach as for ambiguity. The difference is the type of words that can cause multiple functions in a single requirement. For in-stance, connectors (e.g., and) are indicators of a possible lack of atomicity. Another example is the use of multiple imperatives in a requirement (e.g. shall, will, and must).

III. METHOD

Our method for comparing the performance of NLP RE tools consists of three steps: (A) compiling a list of eligible tools, (B) defining a requirement quality standard to which a requirement should conform to and (C) test the performance of each tool against a gold standard. The following sub-sections describe each of these steps.

A. Eligible Tools

We conducted a snowball literature study that led to 50 papers describing tools that apply NLP to requirements [1]. Although many more papers are available on the intersection of NLP and requirements, we pre-emptively excluded all literature that employ outdated approaches, for instance the use of Two Level Grammar in Lee et al. [16], or do not describe a tool. Out of the 50 identified tools, 20 focus on our aforementioned type 1 tool criterion that focuses on finding defects and deviations. The other tools have a different emphasis: 11 of them focus on generating models, 15 on identifying key abstractions, and 4 on other issues. Next, we applied the following inclusion criteria:

* Provides sufficient technical information to understand inner working of the tool (18 remain)

* Accepts NL requirements as input (16 remain)

* Is fully automated – does not require manual pre-processing such as manual POS tagging or creating traceability links (12 remain) * Available for use on the internet or by contacting the author directly (3 remain)

A full overview of all the tools and their reason for exclusion see Table 1 below. The experiment includes the following 3 tools:

1. The Requirements Quality Analyser (RQA) [9] is a commercial tool developed by The Reuse Company. The tool provides morphological, lexical, analytical, and relational indicators.

2. The Tool to InGest and Elucidate Requirements Professional (TIGER-PRO) [14] is an educational tool developed by Joseph Kasser to help students write good requirements.

3. Qualicen [7] is a commercial tool developed by Qualicen GmBH. The tool uses requirement smells to detect quality defects in requirements.

TABLE I. OVERVIEW AND CLASSIFICATION OF NLP RE TOOLS. INCLUDED TOOLS FORMATTED IN BOLDFACE TYPE

Tool Name	Transformation	Automated	Year	Exclusion
Extraction of OLAP req.	Rule/ Ontology Based	Y	2009	Input type
Circe	RB / OB	Y	2006	Not available
NAI	RB / OB	Y	2010	Not available
QuARS	RB / OB	Y	2001	Not available
CRF Tool	RB / OB	Y	2012	Not available
AQUSA	RB	Y	2015	Input type
T1'*	RB	Y	2008	Not available
RAT	RB	Y	2009	Not available
Text2Test	-	-	-	No info
MaramaAI	RB / OB	N	2011	Not auto
EuRailCheck	RB / OB	N	2012	Not auto
UIMA	RB	Y	2009	Not available
DODT	RB / OB	N	2011	Not auto
SREE	RB / OB	Y	2013	Not available
Qualicen	**RB**	**Y**	**2014**	**Included**
Dowser	RB / OB	N	2008	Not auto
RQA	**RB / OB**	**Y**	**2011**	**Included**
Anaphora detection	RB	Y	2011	Not available
Lexior	-	-	-	No info Not available
TIGER-PRO	**RB / OB**	**Y**	**2004**	**Included**

B. Requirements quality standard

Next, we introduce a requirement quality standard to which a requirement should conform, consisting of 16 violation criteria tests. Note that the quality standard only includes those ambiguity and atomicity tests that at least one of the selected tools implements. Each test is accompanied by an explanation and ex-ample based on either the tools' definition or as explained in Berry et al. [2].

T1. “And” – The occurrence of an and that facilitates the description of multiple features.

T2. “Or” – The occurrence of an or that facilitates the description of multiple features.

T3. Multiple imperatives – The occurrence of more than one imperative that facilitates the description of multiple features.

T4. List of items – The occurrence of a list of nouns or noun phrases that facilitates the description of multiple features

T5. Multiple sentences – The occurrence of multiple sentences that facilitates the description of multiple features

T6. Vague pronouns – The occurrence of a pronoun of which it is not certain to which noun or noun phrase the pronoun refers to.

T7. Quantifier – The occurrence of a quantifier of which the scope is not certain.

T8. Unit – The occurrence of a number without a unit.

T9. Slash – The occurrence of a slash of which the nouns or noun phrases on either side have a different meaning.

T10. Non-verifiable terms – The occurrence of terms of which the truth or accuracy cannot be determined.

T11. Subjective – The occurrence of terms which people can interpret differently.

T12. Comparative terms – The occurrence of terms that express a relation to other nouns or noun phrases.

T13. Negative terms – The occurrence of terms that express a constraint.

T14. Superlative terms – The occurrence of terms that signify the greatest form of an adverb or adjective

T15. General ambiguity – The occurrence of general terms that can cause ambiguity.

T16. Escape clause – The occurrence of terms that imply an open ending of a sentence.

C. Gold standard

To test the performance of the tools we aggregated a requirements collection from datasets provided by the developers of all three tools + an internally obtained set of user stories. This way each tool has a baseline for the analysis of the results. Using the requirement standard, the first author of this paper manually pro-cessed the 4 collected datasets. For every instance of a possible defect, for instance a subjective term, the first author determined, based on the requirement standard, whether this possible defect is an actual defect. A requirements expert from Utrecht University (not an author) validated his output.

Next, each of the tools processed these 4 datasets. The difference between the output of the tools and the manually pro-cessed datasets measures the performance of the tools using two main measures: precision and recall [10]. Precision measures how much of the detected errors by the tool are actual errors ac-cording to the manually checked datasets [20]. Recall measures how much of the actual errors according to the manually checked datasets have been detected by the tool [20].

IV. RESULTS AND DISCUSSION

The performance results of the 3 selected tools on the 4 golden standard data sets are available in Table II. Each row reports the precision and recall of a specific feature type for all of the tools. Note that none of the tools implement all possible features. In fact, Qualicen does not include atomicity features at all! The rows with grey highlighting depict average performance for atomicity and ambiguity for each of the tools. The specific score indicates the tool's performance only counting the feature aspects it has implemented, while the overall score averages all relevant features. The precision and recall cell pairs with green highlighting represent the best performing approach according to the tool's recall for that feature. We choose for recall because we agree with Berry et al.'s [3] notion on the criticality of achieving 100% recall of quality defects, sacrificing precision if necessary: The Perfect Recall Condition. The goal of this experiment is to compare the performance of the approaches of the tools. Focussing on precision would not provide us with enough information to compare the performance of the approaches, since precision only looks at which defects a tool does detect, and not at the defects a tool fails to detect. By qualitatively analyzing the output of the tools we learn how and why the experiment results differentiate. In this section we explain how

three types of decisions by tool creators substantially impact the results of the evaluation.

TABLE II. QUANTITATVIE EXPERIMENT RESULTS

Total Score	Qualicen		TIGER		RQA	
	Precision	Recall	Precision	Recall	Precision	Recall
“and”	-	-	0.66	1	0.66	1
“or”	-	-	0.22	1	0.22	1
Multiple imperatives	-	-	1	0.6	0.96	0.96
List of items	-	-	-	-	0.47	0.8
Multiple sentences	-	-	-	-	1	0.56
Atomicity specific	-	-	0.68	0.84	0.68	0.92
Atomicity overall	-	-	0.68	0.65	0.68	0.92
Vague pronouns	0.44	0.25	-	-	0.76	0.91
Quantifier	-	-	0.95	0.67	0.76	0.5
Unit	-	-	-	-	0.1	1
Slash	1	1	1	0.5	1	1
Non-verifiable	0.4	1	0.5	1	0	0
Subjective	1	0.33	1	0.08	1	0.17
Comparative	0	0	1	0.75	1	0.14
Negative	0.83	1	-	-	1	1
Superlative	0.75	1	-	-	-	-
General ambiguity	0.38	0.29	-	-	0.65	0.24
Escape clause	-	-	0	0	1	0.43
Ambiguity specific	0.55	0.36	0.93	0.5	0.65	0.55
Ambiguity overall	0.55	0.26	0.93	0.23	0.65	0.52

1) Different tokenizers

The approaches for detecting conjunctions and and or by TIGER-PRO and RQA are similar: they seemingly both count each instance as a possible atomicity violation. Similarly, all 3 tools detect occurrences of slash-like characters such as / in requirements through pattern matching. Yet, TIGER-PRO only detects half of all slashes. In particular, TIGER-PRO does not detect slashes in between or at the end of words, e.g. passengers/customers and http://. One explanation for this phenomenon is that these tools use different tokenizers to divide a sentence into word-tokens, RQA and Qualicen do split a token on slashes, while TIGER-PRO does not.

2) Dictionary vs. Parsing

Multiple methods are available for detecting vague pronouns and comparative terms. RQA and TIGER-PRO rely on dictionaries of specific bad words. RQA scans for occurrences of 18 pronouns such as it, this and that and TIGER-PRO detects the comparative terms same, also and such as. Qualicen employs a different approach: a part-of-speech tagger which automatically assigns a grammar tag to

each word in a sentence. If a word is tagged as being a pronoun or comparative term, the tool highlights it as ambiguous. Exception rules based on certain types of grammar filters out some pronouns. The downside of this approach is that the 2 most frequent pronouns in the datasets it and this were never tagged as pronouns. For comparative terms, Qualicen employs both POS tagging and morphological analysis, yet it detects no occurrences.

3) The bigger the dictionary the better?

All of the tools utilize dictionaries for multiple feature aspects. Indeed, for both RQA and TIGER-PRO this is the primary method for detecting ambiguity and atomicity violations. The contents of these dictionaries, however, varies substantially. For detecting multiple imperatives TIGER-PRO solely focuses on the imperative shall. RQA also includes the imperatives might, can, will, must and may, resulting in a .36 higher recall at a loss of just .04 precision. Similarly, both TIGER-PRO and RQA use a dictionary to detect ambiguous quantifiers. RQA checks for the quantifiers all, both and any, while TIGER-PRO also includes each. Again, this results in an increase for both precision and recall (.19 and .17). A similar phenomenon applies to the detection of negative term by Qualicen and RQA with 13 and 17 terms. While both obtain 100% recall, RQA obtains a higher precision despite its larger dictionary.

These results suggest that tools with a bigger dictionary generally do better. This is not always the case, however. The sizes of the dictionaries for detecting non-verifiable terms are 7, 13 and 17 for Qualicen, RQA and TIGER-PRO, respectively. Yet, Qualicen and TIGER-PRO obtain 100% recall while RQA fails to detect anything despite its larger dictionary. This result possibly is a consequence of bad luck or bad design choices. It could be bad luck that RQA does not include precisely those five non-verifiable terms that are in the datasets. Furthermore RQA could badly designed considering that it does not include common terms such as measure and brief.

V. NEXT GENERATION TOOL DESIGN

The results of the experiment and their analysis in Section IV serve as a design blueprint for the next generation tool. Together with technical information about the tools provided by their creators, the results enable us to introduce and explain the current superior approaches to the violation criteria tests of the next generation tool design in this section:

T1. “and” – Any instance of and

T2. “or” – Any instance of or

T3. Multiple imperatives – A dictionary with the imperatives shall, might, can, will, must, and may

T4. List of items – Multiple and in a sentence.

T5. Multiple sentences – Multiple “.” that indicate the end of sentence

T6. Vague pronouns – A dictionary with the pronouns him, his, he, it, this, that, their, our, your, my, we, I, they, you, us, her, she, and them

T7. Quantifier – A dictionary with the quantifiers all, both, any, and each

T8. Unit – Any instance of a written or natural number

T9. Slash – A tokenizer that detects the symbol / without space between the words on either side

T10. Non-verifiable terms – A dictionary with the terms affect, all, any, as little as, each, best practice, maximize, maximum, minimize, minimum, quick, rapid, user-friendly, sufficient, include, includes, and including

T11. Subjective – The dictionary of Qualicen with 25 terms

T12. Comparative terms – A dictionary with the comparative terms also, same, and such a

T13. Negative terms – A dictionary with the negative terms nothing, nobody, none, never, doesn't, won't, shan't, mustn't, couldn't, shouldn't, oughtn't, can't, no, nor, non, not, and cannot

T14. Superlative terms – Morphological analysis that detect the greatest form of adverbs and adjectives

T15. General ambiguity – The dictionary of Qualicen with 358 terms

T16. Escape clause – A dictionary with the terms among others, as a minimum, not limited to, not determined, not defined, tbc, tbs, tbd, further, etcetera, etc, and so on, shall be included but not limited to, but not limited to, e.g., eg, example, such as, if possible, if required, possibly, various, when requested by, when required, and to be determined

In theory, a next generation tool that implements these 16 approaches should obtain at least0.92 recall and 0.68 precision for atomicity and 0.76 recall and 0.77 precision for ambiguity. We calculated this by taking the micro average of the performance of the approaches included in the design of the next generation tool [26]. However, far superior approaches might already be available that we cannot include because we base our design on already available tools. After all, our design

is based on tools that are between 13 and 2 years old. For example, it is not inconceivable that new tools employing POS tagging for the detection of vague pronouns outperform the dictionary-based approach we selected for T6. Perhaps a combination of the two approaches is preferable. Furthermore, considering that POS tagging technology is continuously improving it is impossible to predict the exact performance of a next-gen tool. Because of this, next generation tools should undergo a performance test against the same golden dataset used in the experiment in Section III, available online [1].

Nevertheless, one should proceed with caution when starting the development of this next generation tool design. Unfortunately, it is impossible to combine the best-performing features into one tool without recreating the entire technology. In fact, any new custom-built tool released after years of development will likely be yet another tool that has its own strengths and weaknesses in terms of features. As research community, we should start a shared initiative to create a common, modular standard that allows for easily replacing old approaches with superior techniques. The right architecture design could even empower automatic tool assembly with the best modules for a specific context.

An example of a mash-up initiative is the UIMA Framework [24], which enables developers to create sophisticated unstructured information analysis tools by combining application components. The framework defines a standard interface that manages data flow between all available components. Developers can choose to configure existing annotators and repositories, or create new ones for their application. Through component re-use, developers can quickly develop sophisticated tools or web applications that process unstructured text, audio and video. A simplified architecture of this framework is shown in Figure 1.

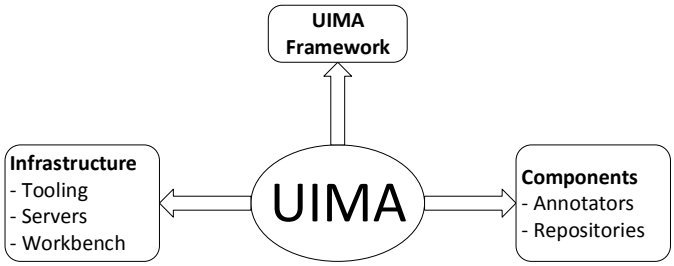

Fig I. THE UIMA FRAMEWORK

Unfortunately, the UIMA Framework requires strong technical capabilities of the user to run it, despite the availability of user guides and tutorials. Achieving widespread adoption requires a user-friendly application with little set-up time or configuration. Ideally, the application exposes a cloud-based interface to allow a wider range of application integrations. It could even built upon existing mashup services such as Zapier1 that enable non-technical users to easily integrate, automate and innovate by moving data between applications.

VI. RELATED WORK

The large number of papers that apply NLP in the context of RE signifies the popularity and relevance of the development of NLP RE tools (see the many tools listed in Section III.A). Yet, few studies exist that rigorously compare the performance of NLP RE tools with one another. Indeed, the majority of papers only describe RE tools and compare their approaches by looking at their implemented features without considering performance [30, 15, 19]. Those that do consider performance, look at varying aspects. For example, [21] compares the recall and precision of three sentence similarity algorithms when applied to requirements and [28] presents a rudimentary comparison of how many occurrences of the words and, or, and/or, all and any three NLP RE tools find. Aside from comparing features, [23] also includes recall and precision scores as reported by the tool authors. The comparative value of these numbers, however, is low.

Although the RE community creates NLP RE tool comparisons on a regular basis, their focus on counting features instead of measuring performance reduces their relevance. We agree with [27]'s call for more rigorous comparisons that report on quality/recall/precision of tools based on a standard benchmark.

VII. LIMITATIONS

The study presented in this paper contains two major limitations. To begin, of the 12 tools eligible for comparison merely 3 were (made) available to us. This has a

negative impact on the comprehensiveness of our study, as well as the expected performance of our next generation tool design. We request tool creators to benchmark their tools using the NLP RE tool performance measurement data set [1].

Additionally, the current golden dataset contains just 112 requirements. Although every feature is included multiple times in this dataset, it is not the case that every dictionary word of each feature is included. This phenomenon might skew the performance measurement in favour of one of the NLP RE tools. Using a bigger dataset will increase the reliability of the performance measurement as well as the quality of the next generation tool

VIII. CONCLUSION AND FUTURE WORK

Although many different NLP RE tools exist, no work is available that compares their performance in detecting ambiguity and atomicity defects in software requirements. In this paper we presented a comparison of NLP RE tools based on a requirement standard consisting of 16 violation criteria for ambiguity and atomicity. The results indicate that none of the tools is clearly superior on all feature aspects. Because of this, we pro-pose a next generation tool design that incorporates all the best features of the previous tools.

Unfortunately, the vast majority of NLP RE tools are no longer available online nor through contacting their creators. There appears to be tendency of creating a prototype for a publication and then abandoning it soon after. We have encountered easily preventable issues such as expired licensing schemes. Be-cause of this, we could only compare the performance of 3 tools.

Moreover, this phenomenon hurts the reproducibility of significant scientific contributions, ultimately hindering the progress of NLP applied to RE [18]. Therefore, we appeal to the community to create a standardized mashup framework or platform to easily assemble feature modules into new NLP RE tools.

Aside from creating such a platform, future work includes replicating this study with more tools and more expansive benchmark data (in other languages). We intend to contribute to and draw from the data provided by the NLRP Benchmark [27].

REFERENCES

The authors would like to thank Henning Femmer for contributing example requirements to the golden datasets, a license for Qualicen and details on its inner workings, the Reuse Company for an RQA license and Fabiano Dalpiaz for commenting on drafts of this paper.

REFERENCES

[1] Arendse, B., & Lucassen, G. (2016). NLP RE Tools data repository. http://www.staff.science.uu.nl/~lucas001/nlp_tools/

[2] Berry, D. M., Kamsties, E., & Krieger, M. M. (2003). From contract drafting to software specification: Linguistic sources of ambiguity. Technical Report, School of Computer Science, University of Waterloo, Waterloo, ON, Canada.M.

[3] Berry, D., Gacitua, R., Sawyer, P., & Tjong, S. F. (2012). The case for dumb requirements engineering tools. Requirements Engineering: Foundation for Software Quality, 211–217.

[4] Boehm, B. W. (1988). Understanding and controlling software costs. Journal of Parametrics, 8(1), 32–68. JOUR.

[5] Brun, E., Steinar Saetre, A., & Gjelsvik, M. (2009). Classification of ambiguity in new product development projects. European Journal of Innovation Management, 12(1), 62–85.

[6] Chantree, F. J., Roeck, A. de, Nuseibeh, B., & Willis, A. (2006). Identifying Nocuous Ambiguity in Natural Language Requirements. Doctoral Dissertation.

[7] Femmer, H., Fernández, D. M., Juergens, E., Klose, M., Zimmer, I., & Zimmer, J. (2014). Rapid Requirements Checks with Requirements Smells: Two Case Studies. Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering, 10–19.

[8] Firesmith, D. (2003). Specifying Good Requirements. Journal of Object Technology, 2(4), 77–87.

[9] Génova, G., Carlos, U., Madrid, I. I. I. De, Génova, G., Fuentes, J. M., Llorens, J., & Moreno, V. (2013). A Framework to Measure and Improve the Quality of Textual Requirements A Framework to Measure and Improve the Quality of Textual Requirements. Requirements Engineering, 18(1), 25–41.

[10] Gill, K. D., Raza, A., Zaidi, A. M., & Kiani, M. M. (2014). Semi-automation for ambiguity resolution in Open Source Software requirements. Electrical and Computer Engineering (CCECE), 2014 IEEE 27th Canadian Conference on, 1–6.

[11] Harwell, R., Aslaksen, E., Hooks, I., Mengot, R., & Ptack, K. (1993). What is a Requirement? INCOSE International Symposium, 3(1), 17–24.

[12] Jain, P., Verma, K., Kass, A., & Vasquez, R. G. (2009). Automated Review of Natural Language Requirements Documents: Generating Useful Warnings with User-extensible Glossaries Driving a Simple State Machine. Proceedings of the {2Nd} India Software Engineering Conference, 37–46.

[13] Kamsties, E., & Peach, B. (2000). Taming ambiguity in natural language requirements. Proceedings of the 13th International Conference on System and Software Engineering and Their Applications, 2, 1–8.

[14] Kasser, J. (2003). Prototype Educational Tools for Systems and Software (PETS) Engineering, (October), 1–11.

[15] Landhäußer, M., Körner, S. J., & Tichy, W. F. (2014). From requirements to UML models and back: how automatic processing of text can support requirements engineering. Software Quality Journal, 22(1), 121–149.

[16] Lee, B., & Bryant, B. R. (2002). Contextual natural language processing and DAML for understanding software requirements specifications. Proceedings of the 19th International Conference on Computational Linguistics, 1, 1–7.

[17] Lucassen, G., Dalpiaz, F., van der Werf, J. M. E. M., & Brinkkemper, S. (2016). Improving agile requirements: the Quality User Story framework and tool. Requirements Engineering, 1–21.

[18] Menzies, T., & Shepperd, M. (2012). Special issue on repeatable results in software engineering prediction. Empirical Software Engineering, 17(1), 1–17. JOUR.

[19] Meth, H., Brhel, M., & Maedche, A. (2013). The state of the art in automated requirements elicitation. Information and Software Technology.

[20] Nakache, D., Metais, E., & Timsit, J. F. (2005). Evaluation and NLP. Database and Expert Systems Applications, 626–632.

[21] Natt och Dag, J., Regnell, B., Carlshamre, P., Andersson, M., & Karlsson, J. (2002). A Feasibility Study of Automated Natural Language Requirements Analysis in Market-Driven Development. Requirements Engineering, 7(1), 20–33.

[22] Pohl, K. (1994). Fifth International Conference on Advanced Information Systems Engineering (CAISE '93) The three dimensions of requirements engineering: A framework and its applications. Information Systems, 19(3), 243–258. JOUR.

[23] Shah, U. S., & Jinwala, D. C. (2015). Resolving Ambiguities in Natural Language Software Requirements: A Comprehensive Survey. SIGSOFT Softw. Eng. Notes, 40(5), 1–7. article.

[24] Sinha, A., Paradkar, A., Kumanan, P., & Boguraev, B. (2009). A linguistic analysis engine for natural language use case description and its application to dependability analysis in industrial use cases. In 2009 IEEE/IFIP International Conference on Dependable Systems Networks (pp. 327–336).

[25] Systems and software engineering -- Life cycle processes --Requirements engineering. (2011). ISO/IEC/IEEE 29148:2011(E), 1–94. article.

[26] Tague-Sutcliffe, J. (1992). The pragmatics of information retrieval experimentation, revisited. Information Processing & Management, 28(4), 467-490.

[27] Tichy, W. F., Landhäußer, M., & Körner, S. J. (n.d.). nlrpBench: A Benchmark for Natural Language Requirements Processing. BOOK.

[28] Tjong, S. F. (2008). Avoiding ambiguity in requirements specifications. DISS, Citeseer.

[29] Wiegers, K. E. (1999). Writing Quality Requirements. Software Development, 7(5), 44–48.

[30] Yue, T., Briand, L. C., & Labiche, Y. (2011). A systematic review of transformation approaches between user requirements and analysis models. Requirements Engineering, 16(2), 75–99.