CHALMERS

ReqCollector: A Versatile Tool for Real-world Requirements Text Analysis

Master of Science Thesis in the Programme Software Engineering and Technology

NIKLAS LINDBLAD
ANDREAS WESTER

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Gothenburg, Sweden. February 2014

ReqCollector: A Versatile Tool for Real-world Requirements Text Analysis

NIKLAS LINDBLAD,
ANDREAS WESTER

Examiner: Richard Berntsson Svensson
Supervisor: Robert Feldt

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden February 2014

Abstract

In large corporations, requirements are often stored in several locations and formats. Projects may use different tools, there may be several systems that are not directly integrated and it is likely that many people are working with requirements across different departments. Therefore, there is a need to collect these requirements to a single location and format. The main purpose of this thesis was to investigate how a general purpose tool using simple lunguistical rules would behave when used on texts in different formats, from different parts of the requirement process. The framework extracts requirements from texts, regardless of format, and formats them according to the users' needs. The framework was developed using design research and was evaluated through a study at Volvo Car Corporation (VCC), where it analyzed three types of requirement documents: a legal text, a structured requirement document and an unstructured requirement document. The results of the study were then evaluated by the authors in cooperation with a representative from VCC. The results show that it is possible to use an automated, general purpose framework for this purpose with good accuracy. In comparison with current state-of-the-art requirement extraction algorithms, the results of this thesis regarding the accuracy of finding requirements are equally accurate, with potential for improvement. The framework can be used by practitioners to collect requirements from different sources and migrate them to a given format.

Keywords: NLP, Requirements, Business Intelligence, Extraction, Feedback

Acknowledgements

The authors would like to extend their gratitude to Professor Robert Feldt of Chalmers University of Technology (CTU) and Mikael Söderholm of Volvo Car Corporation (VCC) for supervising the thesis, and to Dr. Richard Berntsson Svensson of CTU for the examination. Also, the late Assistant Professor Lars Pareto of CTU is thanked for his guidance and kind words. Furthermore, Mr. Erik Wennerbeck and Mr. Andrei Blom provided useful feedback and insights. Finally, the Apache Foundation is thanked for making many of the tools used in the thesis available.

Niklas Lindblad & Andreas Wester, Göteborg 2014-02-02

Contents

1

Introduction

Requirement engineering is an important part of software development, as it lays the foundation for the rest of the development process. Essentially, the result of the requirement engineering phase should ensure that the demands of the stakeholders are met to the highest possible degree in the final product. It is well known that ambiguities, errors or conflicts at this stage may not propagate until much later, at which time the issues may be costly to correct. Also, poorly written requirements documents often lead to project failures as described in Tamai and Kamata (2009).

Tamai and Kamata (2009) shows that if the requirement specification has a consistent level of quality for a number of identified aspects, the project is more likely to succeed. Some of the aspects mentioned by Tamai and Kamata are: how clear the purpose of the specification is, the quality of the individual requirements and definitions, acronyms and abbreviations established. Furthermore, depending on what parts of the specification are of lower quality, different parts of the project are more likely fail. One indication they found was that if the described purpose of the project is of lower quality than the rest of the specification, the project tends to not finish on time.

In many larger corporations, there are issues with managing requirements. Requirements documents in different projects may not have the same format, language and level of formalism, for instance when using both market-driven requirements and business requirements. Analysts may quit, taking knowledge with them. Several people or departments may be working with the same requirements, but in different systems. Taken together, this means that there is a need to gather the requirements to a single location and analyze them against each other. Otherwise, the risk is that even if the requirements are well written, they are conflicting.

One way of extracting and managing requirements that prevent format issues and loss of knowledge is to use a softare solution. Software designed to extract requirements can use structured or unstructured input, where unstructured input is the area explored in this thesis. Unstructured input in this case refers to analysing the contents of a text without any contextual indicators such as headers or sections. The area of analysing text written in natural (unstructured) language is called Natural Language Processing (NLP).

Having found the requirements using NLP, there are also several ways of analysing them. One possibility is to look for similar phrases or words, e.g. finding synonyms based on domain specific dictionaries (ontologies). Another possibility is to create a structural representation of the requirements (e.g. UML) and analysing dependencies between entries to find similarities. The requirements can also be analyzed using formal methods, or by defining a set of rules.

Kof (2005) introduces a semi-automated software tool allowing requirement engineers to create ontologies as a way of identifying domain and context specific synonyms. Several other papers define methods to extract structural representations and relations from natural language requirements, e.g. Joshi and Deshpande (2012) and Shinde et al. (2012).

The goal of this thesis is to evaluate a general purpose tool for extracting requirements. The tool uses a fully automated approach based on simple linguistic rules, and uses no structural information from the texts. The need of such a tool is evident from the conclusions of Mich et al. (2004). According to Mich et al., 79% of the companies targeted in the questionnaire use unstructured natural language in their requirement specifications. Two main potential markets are identified: one consisting of companies wanting to identify user requirements better where the tool could be used as a complement, and another consisting of companies trying to adopt to what Mich et al. calls ”best practices” of software engineering. Also, even though Mich et al. (2004) concludes that the required domain knowledge for handling the requirements is medium to high, they mention that a more general approach is also applicable. This also provides more opportunities to analyze texts from different business areas together. Finally, 64% of the respondents mention that automatisation of identifying user requirements is the most useful mean to improve day-to-day efficiency.

1.1 Purpose

The purpose of this study is to investigate how a fully automated requirement extraction tool based on simple, general purpose linguistic rules performs when used on requirement texts of different types. The accuracy achieved by the tool will determine its usefulness. The study will aim to answer if it is possible to identify and extract requirements using a small set of generic, language based rules and also with what accuracy it is possible to extract the requirements in that way.

1.2 Limitations

The output of the framework will be limited to a database at this point, even though the framework will be able to handle more output types.

The framework will not allow the requirement workers to handle the identification of requirements manually. Automatically analysing the requirements enables processing of much larger quantities of data, and the data can be post-processed by an analyst at a later stage.

Only a small set of requirement documents will be analyzed, and the documents are from the same provider, even though they are very different in nature. This means that the results may not be generalizable to all business domains.

1.3 Structure of the paper

The structure of this paper is as follows; first the background and related works are presented, followed by the methods used to construct and evaluate the solution. Then the technical solution is explained in detail. After that, the results of the study are presented. Finally, the results are discussed and concluded, and some possible continuations described.

2

Background and Related Work

In this chapter, Background and Related work have been merged. The background will describe the areas of NLP and OCR, and the needs of VCC will be explained further. Then related work in the area of NLP will be listed.

2.1 Background

In the elicitation process, the goal is to extract requirement candidate information from the customer. According to Sharma and Pandey (2013), this is done using techniques ranging from surveys, interviews to analysis of textual contracts and agreements. Common for these techniques is that the amount of data gathered can become quite large, and a need for filtering and triage is created in order to narrow the information down to useful requirements. It is understandable that this information is saved in natural language, as it, according to the companies interviewed in Forward and Lethbridge (2002), is the easiest to read and communicate. That the information is often stored as natural language is also supported by Mich et al. (2004), which concludes that 79% of companies use common natural language in their requirement documents.

In Forward and Lethbridge's article it is also presented that more than 50% of companies use plain word processors as their documentation tool. This is further supported by Juristo et al. (2002) where it again is shown that many organizations consider word processors to be their main source of documentation. However, Juristo et al. (2002) also notes that as the number of requirements grow, the disadvantages of the standard word processors begins to show. In some cases, the plain text is enhanced by using some sort of textual structure and using images and links to support the usage of contexts and the underlying logic behind the requirements.

As previously mentioned, the majority of this thesis is concerned with creating an automated software solution to extract requirements from natural language texts, without using any kind of document structure. The requirements are then stored in a local database, for easier querying and integration. The research area of interpreting syntactical and semantical meaning from natural language using software, NLP, is a very close match to the needs of this thesis regarding

the extraction of requirements. There has been previous research on using NLP to automate the requirement process, from elicitation to selection and communication. A few examples of these attempts are Kof (2005), Joshi and Deshpande (2012) and Rago et al. (2013), where NLP is used on a completed requirement specification in order to create UML diagrams, or to find problems with quality attributes. NLP allows the program to look for not only keywords and phrase structure, but also references to other sentences and finding subjects, verbs and objects if necessary. There are no direct alternatives to using NLP, as the area is very broad, but there are several ways of using NLP, as will be shown in section 2.1.1.

2.1.1 Natural Language Processing

Natural Language Processing is based on annotating sentences in texts, in order to extract syntactical and semantical meaning of its constituent parts. This is done in several steps, ranging from individual characters and words to phrases of a sentence. The end result may be represented in a so called parse-tree, as one of the ways suggested by Linckels and Meinel (2011). Another way suggested in the paper is to represent the result as a list. Both representations contain the same amount of information, but the parse-tree representation provides easier access to traversing up and down, while the list is more suited for searching through every node. For the purposes of this thesis, the parse-tree representation will be used, as traversing nodes is important for the flexibility of the program. It is also the representation used natively by the NLP software, as will be explained in chapter 4.2.

A parse-tree is a simple tree-structure representation of the different phrases that make up the sentence, such as verb phrases and noun phrases. An example tree of the sentence “Peter likes pancakes.” is shown in Figure 2.1.

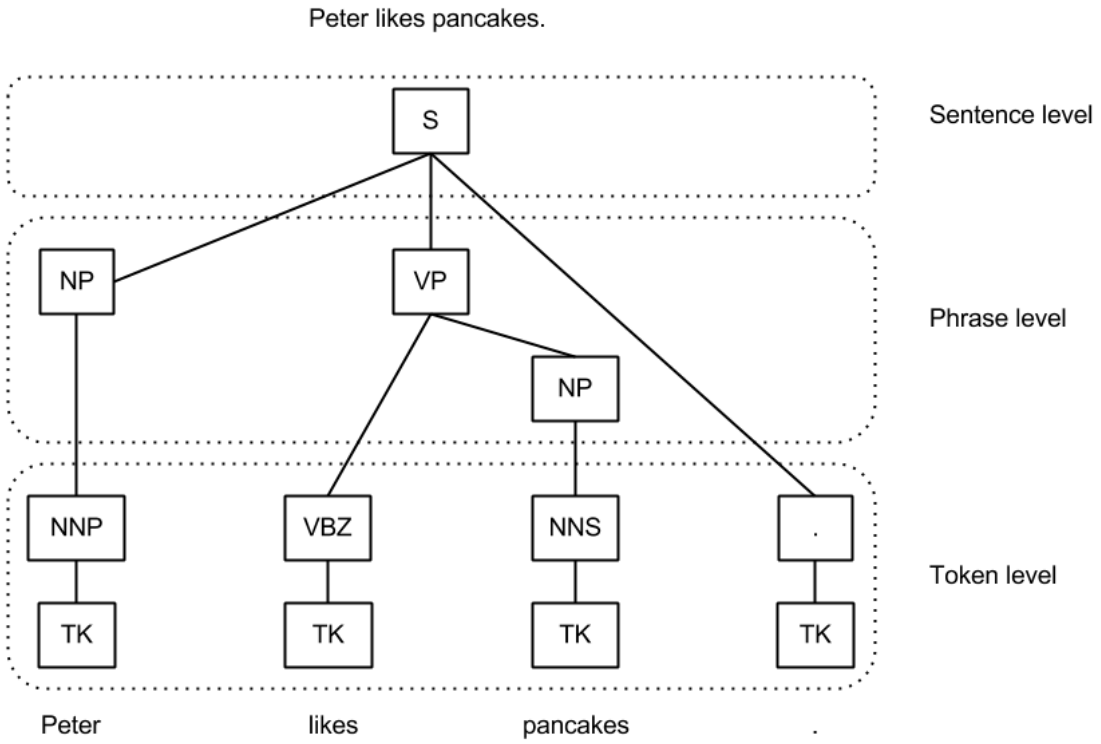

Figure 2.1: An example parse tree. The sentence used is "Peter likes pancakes."

Figure 2.1 shows how the sentence is split into three different levels in the parse tree, where the sentence level denotes the entire sentence and the token level denotes each word or special character. The phrase level is the most interesting, as it is used to find subject, action and object of the sentence, which are used in the tool. Each node has a type that translates to what it denotes, e.g. the NP type translates to a noun phrase. More translations can be seen in the Penn treebank tag set.[1]

There are currently three general approaches to this problem: rule-based, statistical and artificial neural networks according to Collobert et al. (2011). Rule-based NLP is built strictly on grammar, and is the oldest approach of the three. It requires a set of rules for all types of sentence structures, which is easy to extend but very hard to create. There is very little progress in this area in the last years.

Collobert et al. goes on to say that statistical analysis is a common and well-performing approach in modern NLP. It typically uses pre-annotated texts, corpora, to learn what the most common combination of words and sentence structures are. The corpora are often customized for the type of text to process, and are also resource intensive to create. The drawback to this solution is that if a corpora wrongly identifies a structure, it is very hard to correct. There are also statistical solutions that use clustering on similar terms from large, unannotated data sets.

Collobert et al. (2011) means that Artificial Neural Networks (ANN) is a modern, state-of-the-

[1] `http://www.cis.upenn.edu/~treebank/`

art approach that promises both higher accuracy and better performance. It is built on pattern recognition, which in the case of NLP translates to e.g. clustering words into categories. This contextualizes the words in several dimensions using vector representation for likeliness that a word is used in a specific context. Eventually, this can make the words ”money” and ”bank” similar in context, while ”money” and ”monkey” are more grammatically similar. It can also identify that in the sentence ”the car is red”, red is more likely to refer to the adjective than to the verb derived from ”ride”.

2.1.2 Optical Character Recognition

Optical Character Recognition (OCR) is the area of reading text contained in images. The process involves two main stages: feature extraction and classification. Feature extraction deals with identifying characteristics of the symbols found, such as lines in special directions, rounded shapes or dots. These characteristics are then used as input to a classifier that tries to match them against features found in characters. One example would be the using the character ”I”. The feature extraction would identify a single, vertical line. The classification would then find that a single vertical line is used for several characters, such as ”1”, ”I”, ”|” and ”l”, and will decide on one of them as the most likely candidate, depending on the context.

There are many ways to design OCR systems. According to Singh (2013) five of these approaches are: matrix matching, fuzzy logic, feature extraction, structural analysis and neural networks. Matrix matching converts each character into a pattern which is placed in a matrix. This pattern is then compared to an index of characters. Fuzzy logic is a technique that does not evaluate everything to true/false. Instead intermediate values are introduced and the evaluation is done in a more human-like fashion. Feature extraction looks at the specific features of the unknown character and compares it to known features of specific characters. Structural analysis is the same as feature extraction with the addition of comparing sub-vertical and horizontal histograms. Finally neural networks checks the pixel patterns of the unknown character and compares them to known pixel patterns of characters. Currently artificial neural networks currently produce the most reliable results even with damaged characters, as noted in Patel et al. (2012) and Singh (2013).

In order to further show the complexity of producing OCR readings of good quality, consider the following scenarios. For applications where bar codes are handled through OCR, the use case is simple as the algorithms are optimized to look specifically for the presence or absence of bars that represent bits and make up numbers. In this case the bars are the only parameter to consider, and even if they may be skewed or damaged, there is only one way a bar could look.

For applications reading texts from images, the parameters to consider increase dramatically. The number of characters is increased from one to e.g. around 100, counting only the simple characters that can be used in English texts. These include lower and upper case letters, numbers, apostrophes, ampersands etc. Also, different characters are used in different languages. As the characters are based on how characters look, italic and bold text may triple that amount, depending on the algorithm. Considering different font types also multiplies that number by a factor equal to the number of font types supported by the OCR application. Finally, consider that there are characters that resemble each other in many ways, such as ”B” and ”8” and poor quality printed images with broken characters, skewed text, handwritten text etc. and it is easy to see the difficulty in performing good OCR.

A number of factors that are related to the image quality itself, and may heavily affect the

outcome of OCR, are listed below:

- Resolution / DPI
- Font sizes and types
- Contrast
- Text misalignment/skewing
- Text layout
- Discoloration

2.1.3 Industrial case: Volvo Car Corporation

To better understand the needs of larger corporations and to have real data to examine, the authors chose to evaluate progress and the final solution together with Volvo Car Corporation (VCC). VCC is a large and renowned car manufacturer, with more than 420.000 cars sold worldwide in 2012. It is a premium car brand that has become famous for its focus on safety, and models ranging from the small C30 hatchback to the V70 station wagon, S80 sedan and XC90 SUV. The main office is in Gothenburg, Sweden. In 2010, Volvo was bought by the Chinese automaker Geely. Volvo has previously been owned by Ford (1999-2010) and Volvo Group (1927-1999).

The focus of the study is on larger companies in general, where requirement engineering is likely to be an issue. VCC is a good representative of this, as the company is large and the car manufacturing market is heavily regulated. This means that there are many requirements of the cars produced, both internally and externally. Also, they quality for both potential market groups identified by Mich et al. (2004), i.e. companies wanting to identify user requirements better where the tool could be used as a complement, and companies trying to adopt to what Mich et al. calls ”best practices” of software engineering.

The origin of this thesis is VCC having several different systems to capture requirements, but no easy way of sending information between them. Some of the formats of the requirement specifications were considered to include too many unrelated details, which made it hard even for analysts to read them. This in turn made it hard to find connections between requirements, as well as to find possible conflicts and ambiguities that the documents may contain. Therefore, the idea of an automated process that could gather requirements from different sources and export them on a shared format to a single location was created. On top of that, the information would need to be summarized and analyzed, so it could be presented in a user friendly way.

For the setting at VCC, the requirements on the application included that the communication over networks was kept to a minimum, and that the installation would be portable and preferably installation free. The requirement regarding network communication was set to minimize the risk of leaking potentially sensitive data. Requirements for future projects are considered sensitive, and keeping the communication is kept on the local machine rather than over the internet ensures that no information is leaked over the network. Thus, no sniffing attacks or similar can be performed. The portability requirements were introduced mainly so that the program would not have any dependencies and therefore could be moved around between computers easily.

These requirements from VCC limited our possibilities to connect to databases and systems that were not located on the local machine, and therefore narrowed the scope of the tool slightly.

There was no possibility to run the tool on a machine that contained databases or systems, which effectively made documents of different formats the only source to support. This can also be partially justified through arguing that databases and email communications etc. can be exported as files to some extent, and therefore should be able to be analyzed by the tool. Also, because of the portability requirements, a local storage database that required no installation was needed.

2.2 Related Work

In their case study Houdek and Pohl (2000), Houdek and Pohl explore the requirements engineering process at DaimlerChrysler, now known as Daimler. More specifically the assessment was performed on the group tasked with developing the instrument cluster. What Houdek and Pohl found was that the process of elicitation, validation and tracing is not seen as separate steps in a process but are heavily intertwined and are not differentiated to a great extent. Furthermore they suggest that if a similar process assessment is to be made it is worth considering three things. Firstly, how is the whole development process structured. Secondly, what triggers a change in the requirements and how is it handled. Lastly, analyze the documents and use them to get a better understanding of the process.

In Natt och Dag (2005), Johan Natt och Dag et al. present a way of connecting customer wishes to product requirements using linguistic engineering. The first step is to preprocess requirements to remove linguistical differences such as multiplicity and tempus. Then the vector-space model is used as a way of representing the requirements based on the words they contain, and the results are enhanced by using the cosine measure to better identify the similarities between requirements. The paper describes a linguistic approach to finding similarities between requirements, which is distinctly different from the approach this thesis is taking. In Natt och Dag (2005), the requirements are already found and known and the similarities are between sentences. Whereas the approach taken in this thesis is to separate requirements from other sentences.

Kof (2005) describes a way of extracting requirement ontologies, taxonomies and relations based on grouping terms with similar meaning using a human supervised approach. There are differences to this thesis in the approach taken to achieve the goal, where Kof uses an approach where the program is supervised by a human, and this thesis uses an unsupervised approach. However, the method for selecting a subject from a sentence is the same in the paper and this thesis. Kof proves the method effective, and this paper uses the approach.

Shinde et al. (2012) proposes a method for extracting different UML diagrams by building a model of the requirements document and represent it in a Semantic Network. The method of analyzing the text is different from the solution provided by Kof (2005) since it does not involve a parse tree but instead only uses Part Of Speech (POS) tagging and from that extracts the subjects, objects and verbs. This can lead to lower accuracy in some situations, but this is often not the case when it comes to creating UML diagrams, since classes and entities are represented by a single noun most of the time. The similarities between this thesis and the paper are in finding entities and using NLP, otherwise they are completely different.

Much like Shinde et al. (2012),Herchi and Abdessalem (2012) has a method for extracting UML diagrams from requirements. Their solution uses a framework called GATE[2] that is designed to handle unstructured data and through this they get the POS tags but also a syntactic parsing of

[2] `http://gate.ac.uk/`

the sentence. The UML diagrams are created by using an approach that integrates the domain ontology into their solution.

Another way of extracting UML diagrams is suggested by Joshi and Deshpande (2012). What sets their solution apart is that they also use a tool for identifying synonyms and also semantically similar terms. The suggested approach requires external linguistic patterns.

Gervasi and Nuseibeh (2002) uses lightweight formal methods to create partial validation of natural language requirement documents. The approach uses shallow parsing (also called chunking) in combination with structure to validate different aspects of requirements. Gervasi and Nuseibeh further suggest that their solution also leads to styles and quality styles being enforced in the industrial settings that adapt their method. The reason for this is because they are embedded in the tool and thus it supports improved documentation.

A technique is described by Vlas and Robinson (2011) for discovering and classifying requirements in open source projects using an NLP approach and the GATE framework. Although it handles the identification and extraction of requirements, the goal of the approach is to help researchers discover patterns and trends in open source software development. This is done by connecting project success, time lines and quality with certain requirements. This can then be used to give advice about what types of requirements lead to successful open source software.

Rago et al. (2013) use UIMA[3] and NLP to create a tool that a requirements analyst can use to manually find concerns in a requirement document using a querying language. This is a possible extension to the method described in this thesis.

In Shibaoka et al. (2007), a semi-automatic goal-oriented modeling approach using domain ontologies called GOORE is presented. A requirement analyst creates a goal graph, connecting different objects via relations and the underlying automated system suggests things to attach to the graph. The process is similar to creating a UML-diagram with suggested connections and classes. This approach is different to the one suggested in the thesis on many levels, the most similar being that it is eliciting requirements and uses partial natural language processing for suggestions.

Lee and Bryant (2004) convert natural language requirement documents into formal specifications, using XML with DTD specifications created by a user. This is a semi-automated process, and focuses on converting requirement specifications rather than finding requirements, and that is also a main difference to our thesis.

Meth et al. (2013) provides an overview of the related works for this thesis, based on a selection of 36 articles out of 1.126 with "a focus on requirements elicitation and (semi-)automation or the reuse of requirements or knowledge related to requirements". As that is very close to the purpose of this thesis, the result should be a list of the most relevant articles available. Meth et al. (2013) identifies five different areas that are used to classify the solutions, as seen in Figure 2.2. Using that classification, our tool would be classified as: Requirement Identification, Full Automation, Imported Knowledge, Case Study, Correctness & Completeness.

[3] `http://uima.apache.org/`

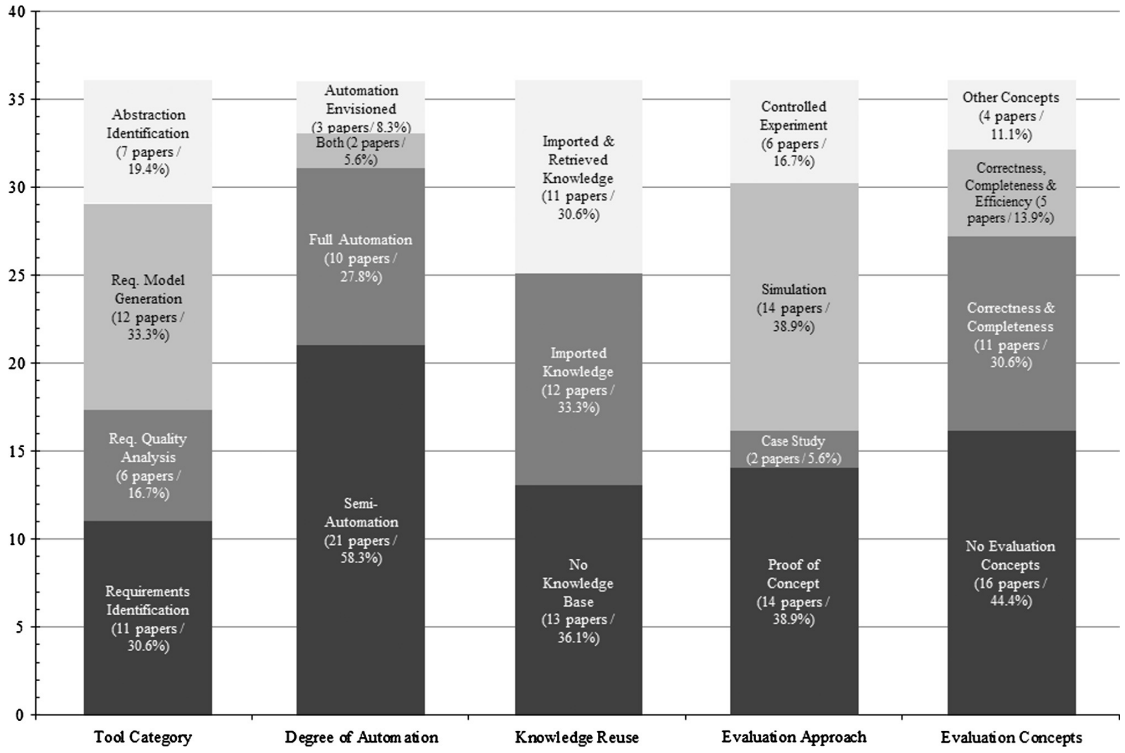

Figure 2.2: The classification of requirement tools by Meth et al.

Apart from Lee and Bryant (2004) and Shibaoka et al. (2007), there are also five other articles mentioned in the category requirement elicitation. Judging by the distribution in Figure 2.2, the categories used are not uncommon but not the most common either. Also, the exact combination of the categories is unique.

There are also commercial products that are similar in that they can take requirements written in natural language and then analyze them for ambiguities. One example of such a tool is Requirement Assistant[4]. It analyzes the document on three different levels; Sentence, paragraph and document level. There are two major differences that set Requirement Assistant apart from the solution suggested in this thesis. Firstly, Requirement Assistant is focused on analyzing the requirement document and not on the extraction of the requirements. Secondly, the solution suggested in this thesis is a minimalistic framework for extracting requirements.

In summary, there are both academic and commercial tools out there that handle similar things to our proposed solution, but the focus on a framework with a minimalistic set of rules that extracts requirements and can produce output on different formats is unique.

[4] `http://www.requirementsassistant.nl/`

3

Method

In this chapter, the process used to develop the software will be explained. It also contains information about, and motivation for, three experiments that were used to determine the usefulness of the software in different settings.

3.1 Study design

The study was conducted using design research during the spring term of 2013. Hevner and Chatterjee (2010) describes design research in detail: how it contributes to science, its differences and similarities to other research methodologies, some best practices, future developments etc. "Design research (DR) consists of activities concerned with the construction and evaluation of technology artefacts to meet organizational needs as well as the development of their associated theories." Hevner and Chatterjee (2010).

The design research was implemented as a custom made agile method, where a typical iteration lasted between two and four weeks. Pair programming was used exclusively, and development items were placed on a virtual board.

At the end of each iteration, the requirements of the software were also evaluated and refined to better follow the progress. In this evaluation step, Mikael Söderholm was used as the main feedback provider from VCC. He was consulted to verify that the progress being made continued to match the requirements of VCC as the software evolved. He is qualified for the task, as he works in the Requirement Team Center of VCC and therefore has insight in the needs of VCC with regards to requirements.

The iterations covered the full length of the project, apart from two separate phases. Before the iterations began, a requirement elicitation phase was used to set up goals and requirements for the project. Detailed information about this phase can be seen in section 3.2. After the project was finished, an evaluation phase began, which is explained further in section 3.5.

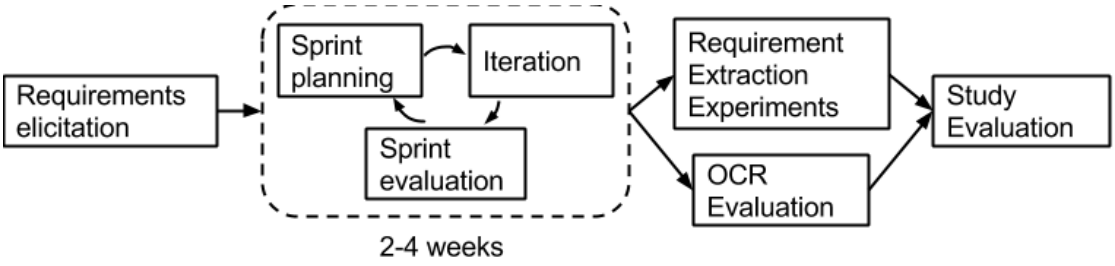

Figure 3.1: The process used to develop and evaluate the software.

The reason for choosing DR for this study was that the final result was supposed to be used in an organization. According to Hevner et al. (2004), DR is more suited for creating artifact that impact people and organizations. Furthermore, Hevner et al. states that DR is more often focused on finding what is effective rather then finding what is true. This also suits the study since the focus is to find a more efficient way of extracting requirements and not prove that extraction of requirements is possible.

3.2 Requirement elicitation

In order to elicit the target requirements for the software solution, a total of three of meetings were held with the supervisor Mikael Söderholm from VCC in the beginning of the study. It was deemed sufficient to include only the supervisor at this point, as he works at the Requirement Team Center of VCC and thereby has some knowledge of the processes and needs of VCC. The intention with collecting requirements from VCC was to ensure that requirements for large corporations were met.

The result of the meeting series, a set of high level requirements, was agreed upon as a starting point. Including the requirements described in section 2.1.3, the requirements are stated below:

- The software must not use any network communication.
- The software should not require installation on the target system.
- The software must be able to read Microsoft Office file extensions and PDFs.
- The software must be able to identify more than 50% of the requirements in the document, with a recommendation of above 70%.
- The software should be able to identify conflicts and ambiguities in the requirements.
- The requirements should be gathered in a single location at the end of the analysis.
- The software should be able to export the requirements gathered.
- The software should not take more than 15 minutes to process a requirement document of around 100 pages.

After the initial requirements were set, the contact with Mikael was held mainly over email. There were also two meetings discussing the progress held in the beginning of April, as well

as the final evaluation held in May. In order to ensure that the requirements initially used were shared with other representatives of VCC, an informal meeting with the section heads of two other departments and several of Mikael's colleagues was held in the beginning of April. The conclusion of that meeting was that the initial requirements were largely correct, with the addition being a specific request to support the handling of legal documents.

3.3 Requirement extraction experiments

Three experiments were conducted in order to show the performance of the tool in different settings. The tool is intended to function independent of how the input data is structured, as it is built exclusively on rules that are based on sentence structure. Therefore, the experiments conducted are of different character both in terms of the level of formalism in the language and in the structure of the document that the data is contained in. All three experiments also contain technical and domain specific terms, which increase the difficulty for the NLP framework to parse the sentences correctly. All experiments use real texts, provided by VCC that have in no way been altered or influenced by the authors unless specifically stated in that experiment.

The software identifies complete requirements, as can be found in requirement specifications. This means that wishes and requests for requirements will not be found as they do not contain the necessary prioritization keywords. This is explained in more detail in section 4. For this reason, the solution was not tested on bug report, support, or feature request data. Also, requirements written on special formats, e.g. use cases, are not always detected.

For the three experiments detailed in section 3.3.1, the desired outcome was to reach at least 90% accuracy for all three types, which would indicate that the software was performing on a level comparable to competing solutions. The accuracy was determined by calculating F-measure (also known as F_1 score or F-score), which is the harmonic mean between two ratios labelled precision and recall. Precision in this case is interpreted as how many of the requirements that should have been captured, that were captured. Recall should be interpreted as how many of the captured requirements were correct. Both are represented as a ratio between 0 and 1. To reach 90% accuracy, the F-measure needed to be at least 0.9. From now on, F-measure, precision and recall will be presented as percentages. For the exact calculation, please refer to section 5.2.1.

For each experiment, the requirements identified were written to a local SQLite database on the format describe in Appendix A, and were later checked manually against the original document in order to calculate the F-measre as described in section 3.5.

3.3.1 Preparing the experiments

In order to test the accuracy of the tool for different inputs, the authors agreed with VCC to test three types of documents. The first type was structured requirement specifications that used e.g. headings, numbering and tables for structure. This type of requirement specification is commonly used, and therefore important to cover. Another tool could use the structural information to achieve good accuracy, and therefore high accuracy for this tool is desired to show its usefulness. Handling structured information without using the structure may also result in additional difficulties, as context may be lost. One example would be using a heading for detailing that a section gives information about testing. This information is lost if the tool does not use the structure.

In order to test this first type of document, VCC provided a structured requirement specification where each section used headings with a predefined order and numbering. Each section had headings for the requirement information, testing procedures, context etc. The language used in the documents was not formal, but the sentences were formulated very clearly to minimize the risk of ambiguity. As this type of document was deemed important, two requirement documents were to be analyzed to cover a larger number of requirements.

The second type of document that was agreed to test was unstructured requirement specifications. This type of specification could be derived from email conversations, meeting protocols, or other written communications such as in forums or incident reports. This type of requirement is also common, but is often included in a structured requirement document at a later stage. Covering this type of document enables the possibility of making sure that the requirements expressed in the structural specification also matches the requirements expressed by users and customers. Some difficulties with this type of documents are that the level of formalism may vary greatly and that there may be several topics included in the same communication making it hard to distinguish which parts are concerned by the requirements.

For this second type of document, VCC provided a requirement specification that only used simple headings. These headings were removed to better simulate an unstructured requirement text. The result is that the document only contains paragraphs containing unstructured requirement texts. The language used is not formal, but the sentences are formulated clearly. Only one document was analyzed.

The third and final type of requirement document to test was a legal text. They contain requirements that are very formal, but also contain large amounts of information that provides context to the requirements. Adding this third dimension enables checking the requirements gathered against legal requirements.

The legal text was a single document provided by VCC, that was only structured through the use of paragraphs.

Based on the characteristics of the documents, it was hypothesized that the tool should perform best on the legal text, where the document contained very formal language. The structured requirement specification was thought to be second best, because of the clearly formulated sentences and structured format. The unstructured document was thought to be the most challenging, as it lacks structure, but the results should still be good as the sentences are formulated clearly. All documents were thought to reach the 90% accuracy mark, indicated by an F-measure of 90% or higher.

3.3.2 Conducting the experiments

The documents to analyze were placed one by one in a folder that the tool uses for input, and then the tool was run. This was to avoid mixing up the requirements, to allow for simpler evaluation. At this point no further analysis was made as the results were to be compared using expert judgment by VCC and the authors in cooperation. The method for evaluating the results is described in section 3.5.

3.4 OCR evaluation

In order to ensure that the OCR functionality works as intended, a short evaluation was made in parallel to the study. As explained in section 2.1.2, there are several factors that influence the performance of OCR software. These factors can, in turn, be combined in various ways to further increase the complexity.

The authors have used three steps in order to create the OCR functionality. First, the document is processed, and images for each page are created. Second, the images are enhanced using an image processing tool. Lastly, the enhanced images are analyzed by OCR software. Open source software has been used for the last two steps, while the first step was created using a GUI automation tool. More details about the solution can be found in section 4. As the focus of this thesis lies in extracting requirements from texts rather than reading all types of documents, only one experiment will be conducted to confirm that the software works as intended.

The limiting factors all affect the OCR performance, but the two previous steps can enhance the result if done correctly. In this case however, it is enough to prove that the end result of the three steps is of sufficient quality, as the functionality as a whole is being evaluated rather than each constituent part.

To best reflect the worst case scenario of the setting that the OCR features are being used in, the document being analyzed should be of lower than average quality. Given the quality of modern scanners and the fact that documents can be rescanned, factors such as resolution, contrast and text misalignment can likely be excluded from the analysis.

The document chosen for analysis can be found on [1]. From that document, pages 2-5 were used, as they contained the largest consequent amount of text with few mathematical formulas or images. This allows the software to work mainly on raw text, which is the area of interest in this case.

The document itself is scanned, and varies in quality. The font is similar to a typewriter, and the characters are close together and bold. There are also several places where the characters are white-out and unreadable, and commas and periods are hard to distinguish between. The contrast varies on each page, although generally it is better in the lower sections of a page. There are small noise spots. The text is slightly misaligned, although the amount varies with each page. The language used in the text is academic English as of 1978, which can be considered modern English.

The factors that are not clearly represented in this document are: poor resolution, font size, text layout and discoloration. Poor resolution in this scenario can be handled through better scanning of the documents. Discoloration is easily solved through a grey scale scanning. Small font size and advanced text layout however, are factors that cannot be altered easily. As the open source OCR tool claims to handle font sizes down to 10 pt. as well as two column layouts, these factors can be ignored because of the focus of the report.

[1] `http://www.princeton.edu/~pkrugman/interstellar.pdf`

3.5 Study evaluation

The study evaluation consists of an overall judgment of both the OCR features and the requirement extraction features of the tool.

The requirement extraction experiments were evaluated through a meeting with our supervisor at VCC, during a one hour long meeting at the 29th of April. At his meeting, the output from the tool in the form of a local database with requirements was manually checked against the requirements in the documents one by one. The definition of a correctly captured requirement that was used includes correctly changing the subject if it references another object, e.g. using the term “it”. However, it does not include capturing all possible context of that requirement, such as preconditions and testing environment. Capturing extra information due to formatting errors, such as missing dots or similar was also ignored.

The results of the requirement extraction experiments were also discussed in terms of how well the expectations set up in the requirements for the tool were met and what implications that had on the future work and on the usability of the tool over all.

The results of the OCR evaluation were intended to be discussed at an earlier meeting with the supervisor at VCC, but as the results (shown in section 5.1) were very promising this was agreed to be unnecessary.

4

ReqCollector

The layout of the chapter is as follows: First, a summary of the tool will be presented. Then, the solution will be explained in more detail starting with tools used and then following the application flow chronologically.

4.1 Application summary

The application consists of three different parts. First, there is a reader that converts the requirement documents into a common format. Secondly, there is an NLP analysis part, where the contents of the documents are read and analyzed in order to extract requirements. During this analysis part, statistics are also gathered. Finally, the output is generated in the form of a list of extracted requirements.

Internally, the tool is built up by a few basic parts. First, the requirement documents are sent through a document reading program. To read a requirement document, the file is put in a specific folder that the program traverses recursively. Essentially, the program looks through the document for any textual content, and the result is returned as a string. If the document type is not supported, or the contents cannot be read, another program that reads text from images is used. The program takes screen shots of each page, and then uses an OCR tool to read the text in the screen shots. This provides support for e.g. scanned documents stored in PDF format. If none of the techniques can read the document, it is skipped and a warning is given to the user.

After reading, the strings are input to the NLP functionality for annotations. The strings with annotations are then fed into the analysis functionality, which uses keywords and a few basic rules to identify requirements. This behavior will be explained more in detail in Section 4.5.

When the analysis is done, the results are shown to the user and stored locally in the database. The flow can be seen in Figure 4.1, and is also structured largely as a typical BI application, as can be seen in Figure 1 of Chaudhuri et al. (2011). The exception is storing the data after analysis, rather than the other way around.

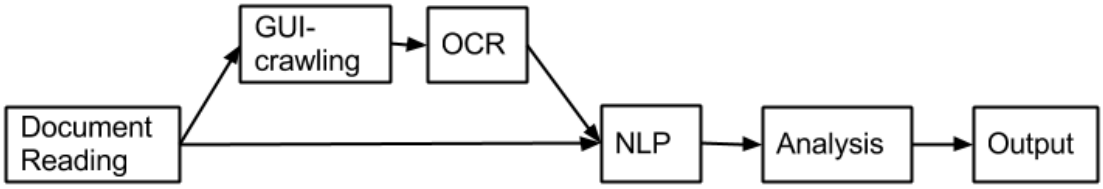

Figure 4.1: The application flow.

4.2 Supporting technologies and libraries

In order to keep the application as portable as possible, as well as being able to use most tools that were investigated for the tasks at hand, Java was selected as a programming language. This provides platform independence and even if it requires that a JVM is installed, it can still be considered portable as there is a JVM available for all major platforms. Using Java was also the best option looking at native language and plug-ins available for the tools, not to mention that both authors have experience in Java coding.

To implement the software there was a need for two major pieces of functionality: extracting text from different document types, and the ability to use NLP on the extracted text. In order to meet these demands, several open source tools and frameworks were used.

To address the first part, Apache Tika[1] was used to extract text from many different document types. Apache Tika is a collection of different open source libraries and can automatically detect which library to use on a certain text document.

In order to handle the second part, retrieving unstructured data from the documents, an NLP solution was needed. There are two major frameworks for handling this: Apache UIMA and GATE. Both helps structure the data and does it in a similar way. UIMA was opted for over GATE because UIMA supports types Wilcock (2009) meaning it can store the information in way that better suits the tool's needs.

To perform NLP on the extracted text Apache OpenNLP[2] was selected. One of the reasons for this is the synergy it has with the overall framework chosen, Apache UIMA. The second reason was because many similar tools uses Apache OpenNLP to analyze text, e.g. Ibrahim and Ahmad (2010), Joshi and Deshpande (2012) and More and Phalnikar (2012), and receives good results.

OCR support was needed to process documents that had been scanned, as well as a way of obtaining the images to use OCR on. For this purpose, an automated GUI testing tool with support for screen shots was used. For OCR, Tesseract[3] was selected. For supporting the OCR software with input, Sikuli[4] was used. Sikuli can provide screen shots of the images, and provides a way of extracting the images easily. The reason Tesseract was chosen for OCR purposes was because it was easy to use and had a built in training module making it possible to train the

[1] `http://tika.apache.org/`
[2] `http://opennlp.apache.org/`
[3] `https://code.google.com/p/tesseract-ocr/`
[4] `http://www.sikuli.org/`

engine in other languages than English. Sikuli was chosen because of the easy integration with Java and the authors had previous experience with Sikuli.

As the setting at VCC did not allow network communication, a local storage solution was needed. A database is preferable in this situation, as it allows for queries both for inserting and updating the data, as well as selecting subsets based on specific parameters. For this task, SQLite was selected. It is lightweight and allows the database to be stored in a local file without any network connection.

4.3 Reading requirement documents

Reading is done primarily using Apache Tika to extract the textual contents of a file to a string. This process is fully automated by the tool, and provides the output directly in Java for most file formats. The main exception to this is image PDF files, which do not contain any textual content but does contain images with text in them. For such documents, OCR is used, and a separate flow is created. At the moment, only PDF files are supported.

4.3.1 Handling images in documents with OCR

First, Sikuli opens the file using Adobe Reader. It then puts the screen of the user in portrait mode to increase the resolution available for the document, and puts the application in full screen mode. Sikuli then takes a screen shot of the current page, which is enhanced by the image processing tool ImageMagick. The enhanced image is then given to the OCR functionality in Tesseract, which reads the image and outputs the results as text. Then, Sikuli scrolls to the next page and repeats the process until there are no more pages, and finally closes the document.

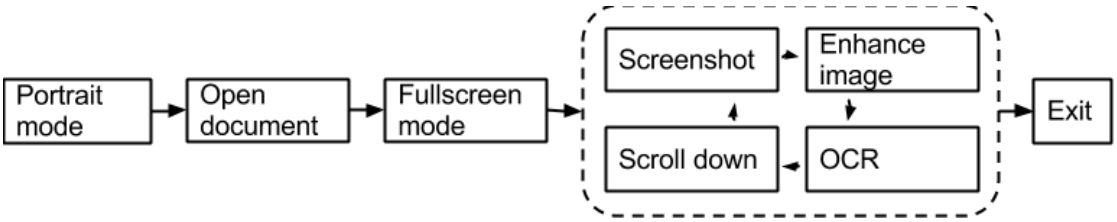

Figure 4.2: The OCR algorithm.

Sikuli captures the image in the highest possible resolution, but cannot affect any other factors that reduce the success rate of OCR. The image processing deskews, scales and anti-aliases the image. ImageMagick could also remove discoloration to some extent, but empirical tests made during the project suggested that this process was handled better by Tesseract, and was left out. Tesseract is then left to work with the remaining factors.

4.4 Annotating requirement texts

Annotators are an essential part of NLP programs, as they create the mark-up of sentence structures that NLP builds upon. To build the parse tree explained in section 2.1.1, several annotators from OpenNLP are used. These include: a sentence detector, a tokenizer, a POS-tagger and a parser. Also, annotators can be customized to find other things, like names and currencies etc. In this thesis, two custom annotators have been implemented to serve the NLP features, as described below.

The first annotator marks keywords that indicate a priority, e.g. “must”, “may” and “should”. These represent the way requirements are separated from other sentences in the method, by claiming that requirements need to be prioritized in order to be valid. A sentence that could be interpreted as a requirement but lacks prioritization is considered a description rather than a requirement, and is omitted from the results. For example, the sentence “The car must have four doors” is distinctly different from “The car has four doors”, where the former is a requirement while the latter is simply a description. This method is also used by Sharma and Kushwaha (2011), and is found to be accurate.

When a sentence uses a priority keyword in a requirement document, it has a very high correlation with a sentence that describes a requirement. Without the keywords, it is very hard to distinguish requirement information from ordinary sentences that may or may not contain requirement information, depending on how the information is interpreted. Including descriptions and similar sentence constructions can be done by the user through modifying the annotator, or including another. Because of the uncertainty, descriptions are not included by default, but since they can be added by the user, the issue is .

The second annotator marks both shorthand and full expressions for units. For instance, the annotator would marks both “2A” and “2 ampere” as containing the unit “ampere”. The annotator uses a few default units as well as a local configuration file, making the list of units customizable for the user. The need for finding units of different types is mainly for filtering and search purposes. For instance, a company may need to update requirements regarding a battery that has switched from 12 to 24 volts. Being able to filter on the unit, as well as other things, therefore helps find the concerned requirements.

4.5 Analyzing requirement texts

After text extraction and NLP annotation has taken place, the method analyzes the data. It uses the data from the requirement annotator to select sentences that contain one or more of the priority keywords. From these sentences, requirement information is then extracted.

First of all, the algorithm divides the requirement sentence into several, simplified requirements if several requirements are described in the sentence. This is done to minimize the risk of ambiguity in requirements. For instance, the sentence “The car must be red and fast” will be split into “The car must be red” and “The car must be fast”.

When the splitting is finished, the algorithm looks for the subject of the sentence. It searches each requirements parse tree left-to-right for a leaf noun phrase directly followed by a verb phrase. That leaf noun phrase is selected as the subject of the sentence, and the method is also confirmed as effective by Kof (2005). All other leaf noun phrases are treated as objects affected

by the requirement. In the example used above, a leaf noun phrase would be ”The car” and a verb phrase would be ”must be”.

After that, the algorithm identifies any units used in the requirement. This is done to be able to search and filter requirements at a later stage. The parse tree is scanned left-to-right for unit annotations. If such an annotation exists, the value of that unit is selected as the value of the requirement. The logic behind this is that if a unit exists in the requirement at this simplified stage, it describes the target value for the requirement. If it does not exist, the algorithm continues the search by looking for adjective phrases, where the adjective phrase that describes the subject is selected. There is also a possibility that none of the previous exist, and no value is selected.

Furthermore, statistics are being gathered throughout the execution, i.e.: how many sentences were in the original document, how many requirement sentences and how many requirements were extracted from those. This is needed as input for the requirement writing process assessment. If the number of requirement sentences differ a lot from the total number of sentences, this is an indication that the format contains overhead information. If the number of requirements gathered is larger than the number of requirement sentences, this is an indication of too much information is likely being saved in requirements and a possibility of ambiguity being present, although it is not a perfect measurement.

Additionally, meta-data is captured from the document files. This is also used as input in the requirement writing process assessment. More specifically; author, last author, date created, last changed, number of versions and word count is used, if existing. If author and last author differs, this likely means that the document is not being handled by a single person, or at least not a single computer. date created, last changed, number of versions and the current time indicate how many times the document has been updated in the timespan. There are several different scenarios available for this data: frequent updates and then nothing, frequent updates until now, infrequent updates then nothing and infrequent updates until now. The scenarios where the last changed date is not recent, this is interpreted as the document being old, and will carry the same significance as the corresponding scenario until now. The scenarios will also be influenced by the statistics gathered for number of requirements.

Lastly, object references are handled. This is done to increase the level of clarity in the requirements, by using the actual object that is referred to instead of ”it” or similar. A requirement that has a subject that references another object, e.g. “it”, is resolved through looking at the requirement to the left in the same sentence for an object or subject that is not, in turn, a reference. If there is no such requirement, the algorithm proceeds to the sentence to the left and solves the same problem until it finds an object or subject to use. This means that the algorithm uses a recursive right-to-left approach. Should there be several possible candidates, the requirement will be flagged for possible ambiguity.

4.6 Framework outputs

There are two outputs from the algorithm. One is a coherent, collected and simplified list of requirements, which is the main output. Another is the statistics and meta-data found in the document.

A more detailed version of the outputs can be seen in appendix A. The first output is a simple

list of requirements. The second output is also a simple list containing statistics and metadata.

5

Results

The performance goal for the framework is to meet the requirements stated in section 3 for all the document types listed. The results will be shown from two main perspectives: OCR accuracy and over all requirement capture rate, according to the experiments described in section 3. Furthermore, the capture rate will be compared to that of similar solutions to show the competitiveness of the framework. Lastly, the results will be compared to the expectations of VCC.

The data presented is anonymous, and only a summary of the findings will be shown in this thesis. For example data that has been created by the authors, see Appendix A.

5.1 OCR accuracy experiment

When evaluating the results of the OCR experiment, all mistakes made by the OCR functionality were counted as errors. This includes e.g. replacing periods with commas, extra line spacing, capitalization of letters and incorrectly identified words in general. As the document quality is at times poor with regard to readability of individual characters, this result should properly reflect the worst case scenario where the software would be considered for use.

The results were evaluated manually by the authors, through comparing the output from the OCR functionality with the document. The resolution of the monitor used was 1680x1050 (portrait mode). The exact output can be seen in Appendix B, and a summary is shown in Table 5.1.

Words	Errors	Accuracy
920	31	99,97%

Table 5.1: Algorithm requirement capture scores

As can be seen in Table 5.1 the accuracy of the OCR reaches 99,97

5.2 Capture rate

True positives is the total amount of requirements correctly captured by the program, while false positives is the number of actual requirements captured that violates the conditions given previously. False negatives denotes the number of requirements that were not captured even though they described an actual requirement. True negatives, which are all the sentences which should not have been captured that were not captured, are omitted. This would mean all sentences that are not requirement sentences, which is not relevant for the analysis. It is also not used when calculating precision, recall or F-measure, which will be explained in chapter 5.2.1. The results of the evaluation can be seen in Table 5.2, Table 5.3 and Table 5.4.

For a more detailed explanation of what is considered a correctly captured requirement, please see section 3.3.2.

File number	Requirements to capture	True positives	False positives	False negatives
1	244	147	5	97
2	328	228	37	100
Average	286	187,5	21	98,5

Table 5.2: Structured requirement document capture scores

Two files were analyzed for the structured requirement specification. The results in Table 5.2 show that there is still a considerable amount of requirements that are not being captured. More about the implications and reasons behind this in section 6.

Requirements to capture	True positives	False positives	False negatives
249	248	7	1

Table 5.3: Unstructured requirement document capture scores

For the unstructured document in Table 5.3, the results are very promising. Very few false positives and negatives were registered, while almost all requirements were captured. This means ultimately that the capture rate is very good.

Requirements to capture	True positives	False positives	False negatives
297	285	0	12

Table 5.4: Legal text capture scores

For the legal document in Table 5.4, the results are similar to the ones for the unstructured document. Only a few false negatives were registered, while almost all requirements were captured. This means ultimately that the capture rate is very good.

5.2.1 Competitiveness

From the gathered data it is possible to calculate the competitiveness of the algorithm. This is done by calculating precision, recall and F-Measure. Precision in this case is interpreted as how

many of the requirements that should have been captured, that were captured. Recall should be interpreted as how many of the captured requirements were correct. F-Measure should be interpreted as a score balancing off precision and recall as equally important, and thus providing the actual value for accuracy over all.

In Table 5.5, Table 5.6 and Table 5.7, Precision, Recall and F-Measure values are calculated using the following formulas:

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives}$$

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

$$F = 2 * \frac{Precision * Recall}{Precision + Recall}$$

Precision	Recall	F-Measure
0,90	0,66	0,76

Table 5.5: Competitiveness analysis of structured requirement document

Precision	Recall	F-Measure
0,97	0,99	0,98

Table 5.6: Competitiveness analysis of unstructured requirement document

Precision	Recall	F-Measure
1	0,96	0,98

Table 5.7: Competitiveness analysis of legal text document

The results of competitive analysis were analyzed by the authors in cooperation with a representative from VCC. When comparing the F-Measure score of 76% in table 5.5 with the requirements from VCC, it is seen that the score matches the requirements both in terms of recommended score and minimum score. This indicates that the algorithm performs at a good level, as the other two experiments show significantly higher values.

Even though the results of the first experiment are acceptable, it was discovered that it is probable that the false positives value can be improved significantly by altering the sentence types captured by the algorithm. This statement is based on terms and structures that were commonly used for the false negatives. In most cases, requirements were missing due to incorrect grammar. To capture those requirements additional keywords could have been added, such as ”min, max, minimum and maximum”. Sentences containing these keywords were found to often describe actual system requirements, but in the documents they lacked prioritization keywords. These errors were also due to the fact that the requirements were often stated in bullet lists, where sentences are often incomplete.

The size of the improvement is estimated to be around a 50% decrease in false negatives. These would also be converted to true positives. As an effect of increasing the number of captured

sentence structures, the number of false positives would likely increase slightly as well. The effects of these suggested improvements are estimated to improve recall by approximately 10% and affect precision a few percent up or down depending on the increase of the false positives, giving a net effect of approximately 4-8% increased F-measure. The issues with bullet lists are handled by the reading program, which would need to be extended or replaced. The suggested improvements were not tested, as they require additional functionality that could not be created during the time span of this thesis. It will however be listed as future work.

Additionally, the result for the unstructured document is very dependent on what kind of language is used. Either the document contains sentences similar to those in a structured requirement document or they are more similar to use cases. If the latter case is true then the result of the algorithm would be more negative than it is in Table 5.6. To improve the score of documents containing use cases more rules regarding the structure of a use case is needed.

Finally, the reason for the high scores from the legal text, as seen in Table 5.7, is the high level of formality in the text. The 12 missed requirements could have been found by adding the keyword ”unlawful” to the algorithm, which is not a prioritization. This could be an indication that the user should be able to add keywords to the algorithm, and that legal texts may have some sentence structures describing requirements that do not use prioritization keywords.

From this it is possible to say that it is very much possible to extract and identify requirements using a small set of generic, language based rules. Furthermore it can be done with a comparably high accuracy.

5.3 Comparing the results to the expectations of VCC

The initial requirements set up for the framework is listed in section 3.2. The actual results are listed below:

- It completed the analysis of the requirement documents in an average of 10 minutes.
- It identified an average of 76% of the requirements.
- It runs on Java, without installation and network communication.
- It finds all requirement information contained in the requirement sentence, but no more.
- It reads all Microsoft Office formats, PDFs, plain text files and more.
- It collects all requirements to one local database, from where the data can be accessed.
- It is able to identify conflicts and ambiguities in the document, to some extent.

The results of the evaluation in general were perceived as useful by VCC. The overall requirement finding score of the algorithm was seen as encouraging, given the room for improvement suggested for the false positives scores. The focus from now on must lie on improving the recall score, as covering as many requirements as possible is vital to making statements about their nature. Also, not capturing all requirements still implies that the documents must be handled manually, using the software as a second opinion.

VCC, as explained in chapter 2.1.3 has experienced an overhead in the format they store requirement information. According to the results of our software seen in section 3.3.1, this overhead exists as 70% of the text is not directly related to requirement sentences. It also indicated that

each requirement stored too much information. Given the ambiguity that is often introduced with writing several requirements in one sentence, this was agreed upon as correct.

Regarding the requirements set up for the software, most requirements were fully met and one was partially met. The partially met requirement is exporting the requirements, which is currently only done to a local database from where they can be read.

In summary, this means that it is possible to identify characteristics of the requirement writing process using the requirements documents, and a fully automated software solution is feasible for the goals of this thesis.

6

Discussion

Two of the experiments showed an excellent result with only marginal room for improvement. The structured requirement document did not show as promising results but still met the requirements set up by VCC, even though it did not reach the anticipated 90% accuracy. The following paragraphs will compare the results to the related works, in order to assess the competitiveness of the solution.

Vlas and Robinson (2011) is closest related to this thesis, where the main differences are the datasets we search and the way we search for requirements. Vlas and Robinson (2011) search the SourceForge dataset for requirement expressed by users. They capture user opinions and wishes based on keywords for beliefs, certainty etc. This differs from the approach used in this thesis, as this thesis searches for requirements based on prioritization, indicating that the expressed opinion is actually expressed like a requirement. The approach used in Vlas and Robinson (2011) covers a more difficult scenario and may include more information than needed, while the solution presented in this thesis focuses on determining what is an actual requirement and may include too little information in some cases. Vlas and Robinson reach an F-measure of 76% for requirements, while we reach an F-measure of 76-98% on an easier set of requirements.

Comparing the results of this thesis to Shinde et al. (2012) is difficult, as they do not present any accuracy measurements. The language processing used by Shinde et al. to support the extraction of classes is simpler than the one presented in this thesis, as it uses chunking (a.k.a. shallow parsing) and stemming rather complete parse trees. The goal of Shinde et al. is also not to use the requirement sentences directly, but to extract entities from them. This fact makes the results even harder to compare, but based on the fact that only chunking is used in Shinde et al. (2012), the results of this thesis should be competitive as full parsing is more accurate.

Herchi and Abdessalem (2012) uses a similar approach for parsing requirement sentences as the one presented in this thesis, although for a different purpose. The purpose of this thesis is to identify requirements, while the approach used in Herchi and Abdessalem (2012) intends to extract entities from already found requirement sentences. Even though the purposes differ, an indication of the accuracy is made in both. Herchi and Abdessalem (2012) reports a precision of 93% and a recall of 83%, giving an F-measure of around 88%. As the F-measure values achieved

in this thesis are 76-98%, with the first experiment having suggested improvements, this gives an indication that the results of this thesis are competitive.

Based on the above comparisons, this thesis like Kof (2005) finds that requirement documents can be automatically analyzed using NLP with competitive accuracy. This thesis also finds that, in this particular case study, the statement still holds true for an approach using a small set of language based rules. The statement is supported by the fact that the suggested solution meets the expectations of VCC, as shown by the results of the three experiments conducted, and by the results of the above comparisons. This indicates that the framework performs at an acceptable level even for the needs of a large company, and should be considered competitive or at least usable.

6.1 Implementation of the algorithm

Because the algorithm used in the framework is based fully based on NLP and does not use any internal structure of the documents analyzed, it can be used on more types of documents. One example of such a usage could be to detect extract information from bug reports. However, as with any NLP approach, the interpretation of syntax and semantics is not flawless and does provide a source for errors. OpenNLP provides two data models for parsing; one based on maximum entropy and one based on linear classification (perceptron). It is possible that choosing another tool for parsing could have generated better results, such as using a tool based on artificial neural networks.

The approach of using keywords to identify requirements does work well for requirement documents according to Kof (2005). This can also be seen in the results in 5.2.1, but used on artefacts like email conversation that contains requirements will likely generate a lot of false positives. The fact that a requirement document is focused on describing the system is the key factor, as email conversation may describe one or several arbitrary subjects at once. Requirements for all of these subjects will then be captured as system requirements, which would be incorrect.

For documents that are more similar to requirement documents, and where the focus is on describing the system, the approach will work in the same way as for requirement documents. The results in chapter 5.2.1 clearly show that the framework works as good, or even better, on legal documents as it does on requirement documents. One important difference however is the fact that in legal documents, sentences that do not contain requirements with prioritization keywords may describe the context in which the requirement is applicable. Unfortunately, this information is not captured by the tool, and will not be shown as errors. This would make the requirement valid but could potentially strip away some of the information. If this is unacceptable the way around it would currently be to have an analyst go through the captured requirement and validate the information gathered. This process should be both easier and less susceptible to human error.

The algorithm also does not capture any description of the system that does not contain a prioritization keyword, as described previously. This means that information that could be interpreted as requirements but lack keywords is not captured. This is a behavior that minimizes the number of false positives found, although it also simply omits some potential requirements.

It is a very computationally heavy process to create a full parse-tree of the text rather than using chunking (also called shallow or light parsing). The choice of doing so is based on the fact that full parsing gives fewer false positives and provides the possibility to use phrases of a sentence.

Fewer false positives is of course desirable, as it provides more accurate data, and phrases can be used e.g. to determine subject, split requirements and to capture quantified values better. Using chunking would on the other hand have generated a program that executed much faster, but as the system is only intended to run when new requirement data is added to the documents, performance is not as important as accuracy.

6.2 Limitations

The evaluation is limited to one company, which may affect how generalizable the findings of this thesis are. In order to verify the results of the thesis, a qualitative approach was chosen. This allowed verifying the results together with experts from the company, and thus providing a more accurate view of the results than a quantitative approach would have. As the method is intended for improving requirements engineering work based on detailed information in requirement documents, accuracy of the results was deemed more important than generalizability. However, the tool is developed to be customizable for the needs of any company and the method itself is based entirely on language specifics. The results should therefore prove generalizable upon further research.

The requirement texts used in this thesis are mainly concerned with describing the area of auto making, although the texts have different characteristics. This may have influenced the results because of the technical nature of the area.

6.3 Implications

In this paper, it is shown that automatically extracting requirements using NLP is feasible, even without using document structure and with a small amount of rules. This means that there can be cost savings for companies from several perspectives. An automated process is much cheaper considering the costs for an analyst manually going through the information, and given the results achieved the tool is close to as accurate as a requirement analyst when dealing with some types of documents. An automated tool also enables much larger quantities of requirements to be analyzed simultaneously. Furthermore, because the rules applied are based on generic language based structures, this tool can analyze many different types of documents from different domains at the same time.

6.4 Future work

In order to generalize these findings, more case studies are needed, and preferably also in different industrial contexts from more business areas. A study to verify the generalizability of the findings quantitatively should be conducted. The additional information gathered regarding the false negatives score should also be investigated to improve the competitiveness of the algorithm.

Since the algorithm is language specific it is valuable if the tool can support as many languages as possible. Currently, the tool only supports text written in English so one of the future

improvements is to support more languages. The reason for the lack of support for more languages is that the accessibility of usable corpora and language models.

To improve the handling finding requirement synonyms, support for determiners and term clustering can be added. Determiners could create links between entities based on multiplicity, and term clustering could be used to identify different terms as referring to the same entity.

Identifying requirement dependencies would add the possibility to analyze the requirement documents for circle references, thus enabling further improvements of requirements.

Finally, giving the user the ability to define what format the requirements are written on would improve the tool even more, e.g. user stories, unstructured or structured. This would both minimize the false positives and also possibly identify more sentences as requirements since the format is known. This would mean that system descriptions, as referred to in Chapter 4 and Section 6.1, could be supported.

7

Conclusion

The goal of this thesis was to evaluate the performance of a general purpose, fully automated requirement extraction tool based on simple linguistic rules. First the level of accuracy that could be achieved with an automated approach was investigated. Then, the tool was evaluated in terms of competitiveness against similar tools and products.

The tool is based on extracting requirements from requirement texts. This was done by first using an open source tool to convert the document formats to plain text, and then applying an NLP solution that read and interpreted the text. The output is a set of collected, formatted requirements.

The accuracy of the tool was measured in three different experiments. Each experiment was characterized by a requirement document type that represented a realistic scenario. The experiments were structured requirement document, unstructured requirement document and legal text, respectively. For each experiment, the accuracy score was calculated using F-measure. The expectation was to reach 90

Over all, the requirements set up by VCC were met to a large extent. The main exceptions include export formats and capturing additional requirement information that may be contained in the document structure. The goal of reaching 90

The results of this thesis indicate that there may be large cost savings for companies in using an automated tool for extracting requirements. The automated tool is cheaper than a requirement analyst going through the documents manually. It also enables large amounts of documents to be processed. Finally, because the proposed tool uses a set of linguistic rules, several different types of documents can be analyzed at the same time.

In order to generalize these findings, more case studies are needed. Also, the tool needs to extend the language support and improve the false negative score for the structured requirement document.

Bibliography

Bucchiarone, A., Gnesi, S., Lami, G., Trentanni, G. and Fantechi, A. (2008), Quars express - a tool demonstration, *in* 'Automated Software Engineering, 2008. ASE 2008. 23rd IEEE/ACM International Conference on', pp. 473–474.

Chaudhuri, S., Dayal, U. and Narasayya, V. (2011), 'An overview of business intelligence technology', *Commun. ACM* **54**(8), 88–98.
URL: *http://doi.acm.org.proxy.lib.chalmers.se/10.1145/1978542.1978562*

Cleland-Huang, J., Settimi, R., Zou, X. and Solc, P. (2007), 'Automated classification of non-functional requirements', *Requirements Engineering* **12**(2), 103–120. Copyright - Springer-Verlag London Limited 2007; Last updated - 2010-07-08; DOI - 1265329481; 35735051; 65792; RQNG; SPVLRQNG7661245.
URL: *http://search.proquest.com/docview/204454143?accountid=10041*

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K. and Kuksa, P. (2011), 'Natural language processing (almost) from scratch', *Journal of Machine Learning Research* **12**, 2493–2537.

Forward, A. and Lethbridge, T. C. (2002), The relevance of software documentation, tools and technologies: A survey, *in* 'Proceedings of the 2002 ACM Symposium on Document Engineering', DocEng '02, ACM, New York, NY, USA, pp. 26–33.
URL: *http://doi.acm.org.proxy.lib.chalmers.se/10.1145/585058.585065*

Gervasi, V. and Nuseibeh, B. (2002), 'Lightweight validation of natural language requirements', *Software: Practice and Experience* **32**(2), 113–133.
URL: *http://dx.doi.org/10.1002/spe.430*

Herchi, H. and Abdessalem, W. B. (2012), 'From user requirements to uml class diagram', *CoRR* **abs/1211.0713**.

Hevner, A. and Chatterjee, S. (2010), *Design Research in Information Systems: Theory and Practice*, 1st edn, Springer Publishing Company, Incorporated.

Hevner, A. R., March, S. T., Park, J. and Ram, S. (2004), 'Design science in information systems research', *MIS Q.* **28**(1), 75–105.
URL: *http://dl.acm.org/citation.cfm?id=2017212.2017217*

Houdek, F. and Pohl, K. (2000), Analyzing requirements engineering processes: a case study, *in* 'Database and Expert Systems Applications, 2000. Proceedings. 11th International Workshop on', pp. 983–987.

Ibrahim, M. and Ahmad, R. (2010), Class diagram extraction from textual requirements using natural language processing (nlp) techniques, *in* 'Proceedings of the 2010 Second International Conference on Computer Research and Development', ICCRD '10, IEEE Computer Society, Washington, DC, USA, pp. 200–204.
URL: *http://dx.doi.org/10.1109/ICCRD.2010.71*

Joshi, S. D. and Deshpande, D. (2012), 'Textual requirement analysis for uml diagram extraction by using nlp', *International Journal of Computer Applications* **50**(8). Copyright - Copyright Foundation of Computer Science 2012; Language of summary - English; ProQuest ID - 1032232384; Last updated - 2012-09-21; Place of publication - Bangalore; Corporate institution author - Joshi, S D; Deshpande, Dhanraj; DOI - 2730112421; 71005152; 126309; CMAP; ICACMAP_CMAP_v50n8d20120101_1051207795-0906.
URL: *http://search.proquest.com/docview/1032232384?accountid=10041*

Juristo, N., Moreno, A. and Silva, A. (2002), 'Is the european industry moving toward solving requirements engineering problems?', *Software, IEEE* **19**(6), 70–77.

Kof, L. (2005), Natural language processing: Mature enough for requirements documents analysis?, *in* A. Montoyo, R. Muńoz and E. Métais, eds, 'Natural Language Processing and Information Systems', Vol. 3513 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, pp. 91–102.
URL: *http://dx.doi.org/10.1007/11428817_9*

Lee, B.-S. and Bryant, B. R. (2004), Automation of software system development using natural language processing and two-level grammar, *in* M. Wirsing, A. Knapp and S. Balsamo, eds, 'Radical Innovations of Software and Systems Engineering in the Future', Vol. 2941 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, pp. 219–233.
URL: *http://dx.doi.org/10.1007/978-3-540-24626-8_15*

Linckels, S. and Meinel, C. (2011), Natural language processing, *in* 'E-Librarian Service', X.media.publishing, Springer Berlin Heidelberg, pp. 61–79.
URL: *http://dx.doi.org/10.1007/978-3-642-17743-9_4*

Meth, H., Brhel, M. and Maedche, A. (2013), 'The state of the art in automated requirements elicitation', *Inf. Softw. Technol.* **55**(10), 1695–1709.
URL: *http://dx.doi.org/10.1016/j.infsof.2013.03.008*

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

More, P. and Phalnikar, R. (2012), 'Article: Generating uml diagrams from natural language specifications', *International Journal of Applied Information Systems* **1**(8), 19–23. Published by Foundation of Computer Science, New York, USA.

Natt och Dag, J. (2005), *Managing Natural Language Requirements in Large-scale Software Development*, Reports on communication systems, Department of Communication Systems, Lund Institute of Technology, Lund University.
URL: *http://books.google.se/books?id=n_ALMgAACAAJ*

Patel, C., Patel, A. and Patel, D. (2012), 'Optical character recognition by open source ocr tool tesseract: A case study', *International Journal of Computer Applications* **55**(10), 50–56. Published by Foundation of Computer Science, New York, USA.

Rago, A., Marcos, C. and Diaz-Pace, J. (2013), 'Uncovering quality-attribute concerns in use case specifications via early aspect mining', *Requirements Engineering* **18**(1), 67–84.
URL: *http://dx.doi.org/10.1007/s00766-011-0142-z*

Sharma, A. and Kushwaha, D. S. (2011), 'Natural language based component extraction from requirement engineering document and its complexity analysis', *SIGSOFT Softw. Eng. Notes* **36**(1), 1–14.
URL: *http://doi.acm.org/10.1145/1921532.1921547*

Sharma, S. and Pandey, S. K. (2013), 'Revisiting requirements elicitation techniques', *International Journal of Computer Applications* **75**(12). Copyright - Copyright Foundation of Computer Science 2013; Last updated - 2013-09-13.
URL: *http://search.proquest.com/docview/1428959902?accountid=10041*

Shibaoka, M., Kaiya, H. and Saeki, M. (2007), Goore : Goal-oriented and ontology driven requirements elicitation method, *in* J.-L. Hainaut, E. Rundensteiner, M. Kirchberg, M. Bertolotto, M. Brochhausen, Y.-P. Chen, S.-S. Cherfi, M. Doerr, H. Han, S. Hartmann, J. Parsons, G. Poels, C. Rolland, J. Trujillo, E. Yu and E. Zimányie, eds, 'Advances in Conceptual Modeling – Foundations and Applications', Vol. 4802 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, pp. 225–234.
URL: *http://dx.doi.org/10.1007/978-3-540-76292-8_28*

Shinde, S. K., Bhojane, V. and Mahajan, P. (2012), 'Nlp based object oriented analysis and design from requirement specification', *International Journal of Computer Applications* **47**(21). Copyright - Copyright Foundation of Computer Science 2012; Last updated - 2012-12-11; DOI - 2837706481; 74158012; 126309; CMAP; ICACMAP_CMAP_v47n21d20120101_1051207475-0574.
URL: *http://search.proquest.com/docview/1233444203?accountid=10041*

Singh, S. (2013), 'Optical character recognition techniques: A survey', *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)* **2**(6), 2009–2015.

Tamai, T. and Kamata, M. (2009), Impact of requirements quality on project success or failure, *in* K. Lyytinen, P. Loucopoulos, J. Mylopoulos and B. Robinson, eds, 'Design Requirements Engineering: A Ten-Year Perspective', Vol. 14 of *Lecture Notes in Business Information Processing*, Springer Berlin Heidelberg, pp. 258–275.
URL: *http://dx.doi.org/10.1007/978-3-540-92966-6_15*

Vlas, R. and Robinson, W. (2011), A rule-based natural language technique for requirements discovery and classification in open-source software development projects, *in* 'System Sciences (HICSS), 2011 44th Hawaii International Conference on', pp. 1–10.

Wilcock, G. (2009), 'Linguistic processing pipelines: Problems and solutions'.

A

Appendix A

The output presented in this appendix is based on example data created by the authors.

Input data

The data below is used as input through a .txt file and generates the outputs presented in the sections following the current.

- The car must be red. It must be fast. It must also be convertible.
- The cat must be white, the car must be red. It must not be brown.
- The dog jumped. It must be flying.
- The zebra was black and white and the antelope jumped higher than the dog. It must have been scared, I guess.
- The cat must be black.

Statistics output

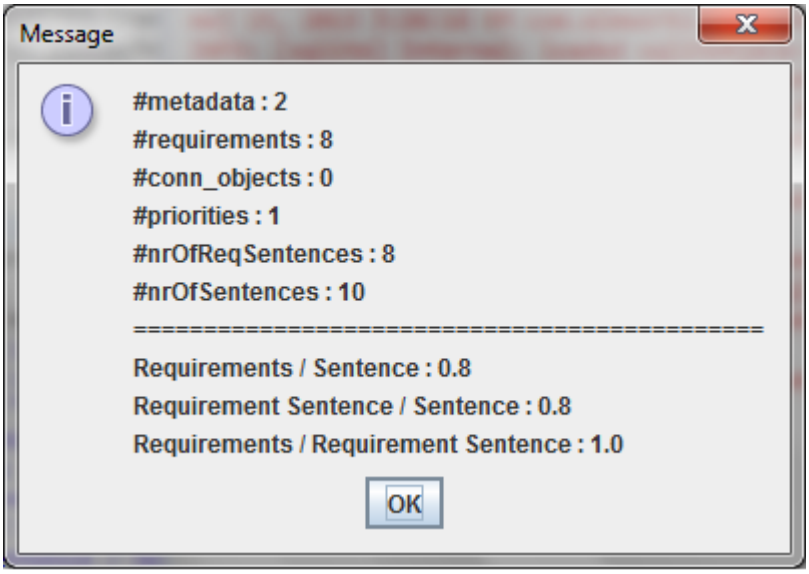

Figure A.1: Example output of statistics

The statistics shown in Figure A.1 in this case are not very interesting, as they reflect only ten sentences, but it is clear that two sentences do not contain requirements.

Ambiguity output

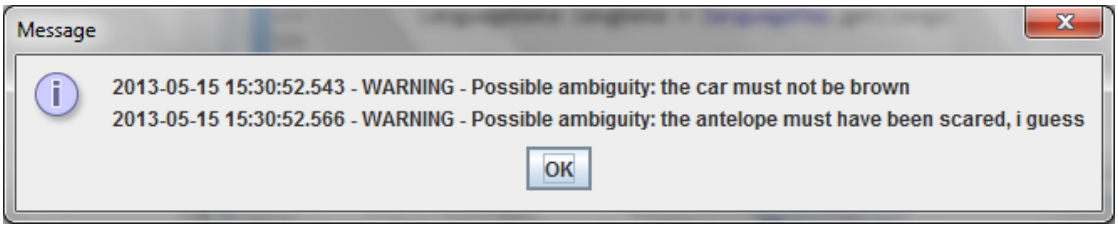

Figure A.2: Example output of ambiguity

The ambiguities in Figure A.2 are both correctly identified, as the sentence before contains several possible candidates for subject and the original sentences both start with the term “it”.

Conflict output

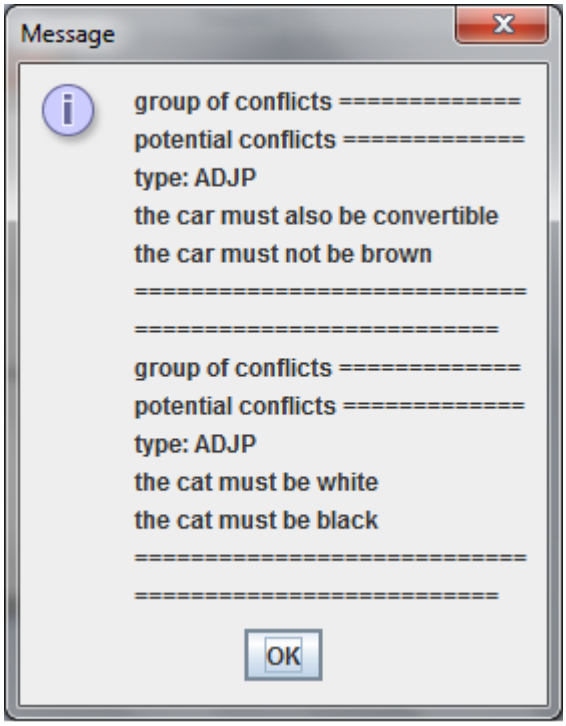

Figure A.3: Example output of conflicts

The potential conflicts in Figure A.3 are split into two groups because they handle different subjects. Each conflict is based on the subject being same and the values being of the same type, but different. In the first case, the flagging is likely incorrect as there is no clear error in being both convertible and not brown. In the second case, the possibility for conflict is higher. It is likely that the cat should not be both black and white, although it might be striped or multi-colored. From the information available, it is impossible to tell if the flagging is truly incorrect or not. The algorithm also does not have any notion of colors vs. other adjectives, making the flagging more common in the first place.

Database output

Important to note in Table A.1 is that the missing values all are considered verbs or adverbs by the OpenNLP parser, which is not supported in the algorithm. Except for the first entry, this is also how the authors would interpret the sentences. The term “red” should have been tagged as an adjective rather than a verb.

id	value	valueType	formatted
1	null	null	the car must be red
2	null	null	the car must be fast
3	convertible	ADJP	the car must also be convertible
4	white	ADJP	the cat must be white
5	brown	ADJP	the car must not be brown
6	null	null	the dog must be flying
7	null	null	the antelope must have been scared, I guess
8	black	ADJP	the cat must be black

Table A.1: Database output

Characteristics output

Because the example file is in plain text format, some of the values used for characteristics are not applicable. Author and versions are not available, and therefore the two measurements that use these variables are excluded.

Formula	Value	Status
(#Ambiguities + #Conflicts) / #Requirements	1/2	Warning
#Requirement sentences / #Sentences	1	OK
#Requirements / #Requirement sentences	1	OK

Table A.2: Characteristics calculations.

This would give the text output ”Too many issues per requirement”.

B

Appendix B

This appendix contains the results of the OCR accuracy experiment. Any errors present in the document are indicated with "((err))" after the word in question, or in the white space that should not have been included. In most cases, these errors can be traced back to the quality of the document. For instance unreadable characters being interpreted incorrectly cannot be blamed on the OCR software, even though an internal dictionary might have resolved some of the issues.

Abstract

This paper extends interplanetary trade theory to.en((err)) interstellar setting. It is chiefly concerned with the following question: how should interest charges on goods in transit be computed when the goods travel at close to the speed of light? This is a problem because the time taken in transit will appear less to an observer travelling with the goods than to ((err)) a stationary observer. A solution is derived from economic theory, and ((err)) two useless but true theorems are proved.

I. Introduction

Many critics of ionventional((err)) economics have argued, with considerable Justification((err)), that tie((err)) assumptions underlying neoclassical theory bear little resemblance to the world we know. These critics have, however, been too quick to assert that this shows that mainstream economics can never be of any use. Recent progress in the technology of space travel, as well as the prospects of the use of space for energy production and colonize ion((err)) (O'Neill l976} make this assertion doubtful; for they raise the distinct possibility that we may eventually discover or construct a world to which orthodox economic theory applies. It is obvious, then, that economists have a special interest in understanding, and, indeed, in promoting the develop- ment of an interstellar economy. One may everihope that formulation of adequate theories of interstellar economic relations will help accelerate the emergence of such relations. Is it too much to suggest that current work might prove as influential in this development as the work of Adam Smith was in the initial settlement of Massachusetts and Virginia?

This paper represents one small step for an economist in the direction of s((err)) theQy'Of((err)) interstellar trade. It goes directly to the problem of trade over stellar distances, leaving aside the analysis of trade within the Solar System. Interplan etary((err)) trade, while of considerable empirical interest (Frankel 1975) raises no major theoretical problems, since it can be treated in the same framework as interregional and international trade. Among the authors who have not pointed this out are Ohlin (1933) and Samuelson (1957). ((err)) Interstellar trade, by contrast, involves wholly novel considerations. The ((err)) most important of these are the problem of evaluating capital costs on goods

[U((err))

in transit when the tine taken to ship them depends on the observer';((err))

reference frame; and 'be((err)) proper modellin;((err)) of arbitrage in interstellar capital markets where — or whzn((err)) (which comes to the same thing) — sinultaneity((err)) ceases to have an unambiguous meaning.

These complications make the theory of interstellar trade appear at first quite alien to our usual trade models; presumably it seems equally human to alien trade theorists. But the basic principles of maximization and opportunity cost will be seen to give clear answers to these questions. I do not pretend to develop here a theory which is universallv((err)) valid, but it may at least have some galactic relevance.

The remainder of this paper is, will be, or has been, depending on the reader's inertialframe, divided into three sections. Section II develops the basic Einsteinian framework of the analysis. In Section III this frame- work is used to analyze interstellar trade in goods. Section IV then considers the role of interstellar capital movements. It should be noted that, while the subject of this paper is silly, the analysis actually does make sense. This paper, then, is a serious analysis of a ridiculous subject, which is of course the opposite of what is usual in economics. II.

Fundamental Considerations

There are two major features distinguishing interstellar trade from the interplanetary trade we are accustomed to. The first is that the time spent in transit will be very great, since travel must occur at less than light speed; round trips of several hundred years appear likely. The second is that, if interstellar trade is to be at all practical, the spaceships ((err)) which conduct it must move at speeds which are reasonable fractions of the ((err)) speed of light.

Becnuse((err)) intcrstelinr((err)) trade will take so long, any decision to launch ((err)) a cargo will necessarily be a very long-tern((err)) investment project, and would hardly be conceivable unless there are very extensive futures markets. I will assume, then, that future futures markets are, well, futuristic in their development. In fact, I will assume that investors, human or other-vfie((err)), are able to make perfect forecasts of prices over indefinite periods.

The second feature of interstellar transactions cannot be so easily dealt with (physicists are not as tolerant as economists of the practice of assuming difficulties away). If trading space vessels move at high velocities, we can no longer have an unambiguous measure of the time taken in transit. The time taken by the spacecraft to make a round trip will appear less to an observer on the craft than to one remaining on Earth. Since an interstellar voyage is an investment project which must have a positive present value, there is obviously a problem in deciding which transit time to use in the present value calculation.

This is an inertial problem - which becomes a weighty problem in a gravitational field - requiring an economic analysis, provided in the next section. In this section I develop the necessary physical concepts, il- lustrated in Figure I. Consider trade between two planets, Earth and Trantor. I assume that the two planets may be regarded as being in the same inertial frame. Then their world lines in space-tine can be represented by two parallel lines, shown as EE'E" and TT'T" in the figure. Several types or contact between the two planets are also shown. The line ET is the world line of an electromagnetic signal - say, a rerun of Star Trek - ((err)) sent from Earth to Trsntor((err)). If time is measured in years and space in ((err)) light years, ET will have a h5—degree((err)) slope. The line E'T'((err)) is the world-