UNIVERSITY OF OULU

FACULTY OF INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING

Lilli Hiltunen

REDUCING STRUCTURAL AMBIGUITY IN NATURAL LANGUAGE SOFTWARE REQUIREMENTS SPECIFICATIONS

Master's Thesis
Degree Programme in Computer Science and Engineering
September 2020

Hiltunen L. (2020) Reducing Structural Ambiguity in Natural Language Software Requirements Specifications. University of Oulu, Degree Programme in Computer Science and Engineering. Master's thesis, 92 p.

ABSTRACT

The ambiguity of natural language (NL) causes miscommunication and misunderstandings. Precision of language is particularly important in software development when handling requirements agreed between the customer and the provider. Software Requirements Specification (SRS) is a commonly used document type for specifying the requirements. A strict standard for how every SRS should be constructed does not exist, and thus it is often written in NL. However, some restricted languages can be used for specifying requirements. An example of such is Easy Approach to Requirements Syntax (EARS).

In this thesis is presented an automated tool for reducing the structural ambiguity of requirements by converting NL into EARS form. Four different text datasets were used for testing the converter and they were compared before and after conversion and against each other. Both performance and ambiguity reduction of the tool were assessed using various measures. Since a standard ambiguity measurement was not available, a combination of sentence structure assessment, word occurrences against Zipf's law, readability score and information complexity was used.

The results suggest that the tool reduces structural ambiguity of sentences. The tool is successful in converting NL into the different EARS patterns and the converted sentences are less complicated and more readable, according to the results. This hints at the possibility of creating more automated tools that could be used to reduce ambiguity in NL SRS. It might not be possible to make people start using a restricted language, like EARS, for writing the documents, but with the help of automated converters, sentences could be mapped to more restricted forms to help with making better sense of them.

Keywords: Natural Language Processing, Ambiguity Reduction, Requirements Engineering, Software Requirements Specification, Easy Approach to Requirements Syntax (EARS)

Hiltunen L. (2020) Luonnollisen kielen rakenteellisen moniselitteisyyden vähentäminen ohjelmistojen vaatimusten määrittelyissä. Oulun yliopisto, tietotekniikan tutkinto-ohjelma. Diplomityö, 92 s.

TIIVISTELMÄ

Luonnollisen kielen epämääräisyys aiheuttaa vaikeuksia kommunikoinnissa ja ymmärtämisessä. Kielen tarkkuus on erityisen tärkeää ohjelmistokehityksessä silloin kun käsitellään asiakkaan ja tarjoajan keskenään sopimia vaatimuksia ohjelmistolle. Ei ole olemassa tiukkaa standardia sille miten vaatimusten määrittelydokumentti pitäisi rakentaa, joten se usein kirjoitetaan luonnollisella kielellä. Siitä huolimatta joitain rajoitettuja kieliä voidaan käyttää yksittäisten vaatimusten määrittelyyn. Eräs esimerkki rajoitetusta kielestä on Easy Approach to Requirements Syntax (EARS).

Tässä diplomityössä esitellään automatisoitu työkalu vähentämään rakenteista epämääräisyyttä muuttamalla luonnollista kieltä EARS-muotoon. Neljää erilaista tekstiä käytettiin työkalun testaamiseen ja niitä verrattiin toisiinsa sekä ennen että jälkeen muuntamisen. Työkalun toimintaa ja epämääräisyyden vähentämistä mitattiin useilla metriikoilla. Epämääräisyyden mittaamiseen valittiin joukko kvantitatiivisia metriikoita: lauserakenteita analysoitiin, sanojen ilmiintyvyystiheyttä ja lausiden luettavuutta mitattiin ja informaation kompleksisuuttakin verrattiin muunnettujen ja muuntamattomien tekstien välillä.

Tulosten perusteella esitelty työkalu vähentää lauseiden rakenteellista epämääräisyyttä. Se muuntaa onnistuneesti luonnollista kieltä EARS-muotoon ja tulosten mukaan muunnetut lauseet ovat vähemmän monimutkaisia ja luettavampia. Tämä viittaa siihen, että automatisoiduilla työkaluilla voisi olla mahdollista vähentää epämääräisyyttä luonnollisella kielellä kirjoitetuissa vaatimusten määrittelydokumenteissa. Vaikkei ihmisiä saataisikaan kirjoittamaan vaatimusten määrittelyjä rajoitetuilla kielillä, automatisoiduilla kielen muuntajilla lauseita voidaan uudelleenmuotoilla rajoitetumpiin muotoihin, jotta niistä saataisiin paremmin selvää.

Avainsanat: Luonnollisen Kielen Käsittely, Epämääräisyys, Moniselitteisyys, Monitulkinnallisuus, Monimerkityksellisyys, Vaatimusten Määrittely, Ohjelmistojen Vaatimusten Määrittely, Easy Approach to Requirements Syntax (EARS)

TABLE OF CONTENTS

FOREWORD

I would like to thank prof. Mourad Oussalah for acting as the principal supervisor of this thesis and Kari Miettinen and Sami Hokuni for their technical supervision. I would also like to thank Marjo Kauppinen and Eero Uusitalo for their requirements engineering and EARS expertise. Thank you Mikko, Janne, Janne,[1] Jussi and Markku for your enthusiasm and general support during working on this project. Last, but not least, thanks go to Antti and Mikki for their mental support.

Since I originally wrote this thesis in 2018 and I have hardly made any changes since then, I am very thankful that I got the chance to publish it after all this time. So many thanks, everyone.

Vantaa, Finland September 8, 2020

Lilli Hiltunen

[1]Yes, two Jannes

ABBREVIATIONS

ACE	Attempto Controlled English
AI	Artificial Intelligence
ARI	Automated Readability Index
BDT	Branching Direction Theory
CCG	Combinatory Categorical Grammar
CNL	Controlled natural language
EARS	Easy Approach to Requirements Syntax
FIDE	Fédération Internationale des Échecs (World Chess Federation)
FN	False negative
FP	False positive
FSA	Finite-state Acceptor
FST	Finite-state String Transducer
IE	Information Extraction
IEEE	Institute of Electrical and Electronics Engineers
IR	Information Retrieval
LZMA	Lempel-Ziv-Markov chain algorithm
ML	Machine Learning
NER	Named Entity Recognition
NL	Natural Language
NPC	Noun Phrase Chunk
NLP	Natural language processing
NLU	Natural language understanding
OS	Operating system
OWL	Web Ontology Language
POS	Part of Speech
RMS	Requirements management system
RE	Requirements engineering
RS	Requirements specification
SCT	Semantic Classification Tree
SRS	Software Requirements Specification
TN	True negative
TP	True positive
UI	User Interface

EARS Patterns

EDP	Event-Driven pattern
OFP	Optional Feature pattern
SDP	State-Driven pattern
UP	Ubiquitous pattern
UBP	Unwanted Behaviour pattern

Tree form

ADJP	Adjective phrase

CD	Cardinal number
Det	Determinant
IN	Preposition or subordinating conjunction
JJ	Adjective
MD	Modal
N	Noun
NN	Singular noun or mass
NNP	Singular proper noun
NNS	Plural noun
NP	Noun Phrase
POS	Possessive ending
PP	Prepositional Phrase
S	Sentence
SBAR	Subordinate Clause
V	Verb
VB	Verb in base form
VP	Verb Phrase
VBN	Past participle verb
VBP	Singular present verb (not 3rd person)
VBZ	3rd person singular present verb

1. INTRODUCTION

In software engineering, a software requirements specification (SRS) is the document that defines the software and its functions. The software designer(s) and the commissioner have to agree about its contents. Unfortunately SRS is often written in natural language (NL). NL can be very ambiguous, but creating tools that clarify the NL facilitates reducing ambiguity in SRS documents. [1]

Ambiguousness can lead to disagreements about the meaning of the requirements specifications. At worst, ambiguity can be disastrous for the software project, since it causes delays and additional work and costs. Building the system becomes the more difficult the later the ambiguities are resolved in the project. When the ambiguities are found, it requires fixing them in the specification and in the product, which can be expensive. If the ambiguities are not found before the product is finished, the customer may be very unhappy with the product, since it may not be what the customer expected.

Using controlled natural languages (CNL) can solve some of the ambiguousness by eliminating some of the structural and semantic ambiguity. Applying a CNL requires training and, in some cases, a system that enforces their usage. Training and the system can be expensive. CNL is a lot more restrictive than NL. Some CNLs only use a certain set of words and some allow only some types of sentence structures. [2]

However, it has been proven that it is easier to extract the logic behind a system when the requirements are written using CNL rather than NL. [3] If the NL SRS is first transformed into CNL SRS, automating the process of transforming NL SRS into a different formal form might become easier. This in turn could reduce more ambiguity.

Some tools already exist for keeping track of the ambiguity in SRS documents. They typically focus on analysing the sentences against each other or they may force the writer to fill in the blanks of language templates. [4, 5, 6] Some level of converting NL sentences into more structured formats exist [7, 8, 9]. But an unsupervised tool that transforms NL sentences into unambiguous requirements has not been created yet.

Easy Approach to Requirements Syntax (EARS) is a CNL used to write requirements in a less ambiguous way than just writing them in NL. [10] EARS does not restrict the vocabulary, but it contains templates in which the text should be fitted into. Fitting NL in templates creates a more unambiguous structure for the language without taking away all of its expressiveness and flexibility.

In this thesis is presented the EARS Converter[1] – an automated tool that transforms NL requirements into their EARS equivalents. The Converter reduces the structural ambiguity of the sentences. Because the Converter's structure is simple, it could be integrated to many different systems and processes. In its current form, it is a tool for a requirements engineer to get suggested EARS form equivalents of NL requirements.

The Converter is assessed with a group of measures chosen for determining the structural ambiguity of the input and output data. The datasets chosen for the study vary in their content and format to bring out how the Converter handles diverse SRS data. All of the measures are quantitative and the datasets are freely available.

Creating automated tools that lessen the ambiguousness of a text also paves the way for natural language understanding (NLU), which is machine's automatic comprehension of NL. [11] Better NLU could help different artificial intelligences (AI) to reach

[1]There is more about the EARS Converter in the Implementation chapter.

more humanlike comprehension. With SRS documents, it can lead to AIs that could understand the SRS and translate it to different forms to make the SRS easier to understand by humans. It could also lead to AIs that could produce working code by interpreting the SRS documents correctly.

2. BACKGROUND

In requirements engineering, Requirements Specification (RS) is one of the most important documents, and in software requirements engineering the equivalent is the Software Requirements Specification (SRS). A problem with SRS is that the specifications are often written in natural language (NL). [12]

NL is ambiguous by nature. To reduce ambiguity in SRS documents, a method of turning NL into a controlled natural language (CNL) was developed during this thesis. CNLs are specifically developed for reducing the ambiguity of NL, and the chosen target CNL, Easy Approach to Requirements Syntax (EARS), was built specifically for RS. [10]

Structural ambiguity was chosen as the main type of ambiguity to reduce for this thesis. To compare structural ambiguity of texts, a few measures of structural ambiguity were gathered together. Since a standard structural ambiguity measure does not exist, most of the chosen methods are indirect ways to measure ambiguity.

In this chapter is explained some background information for the methods chosen for reducing ambiguity of natural language. In the first section, some basic concepts of NL and NLP are presented. In the second section ambiguity and complexity are reviewed in the context of NL. In the third section the features of SRS are discussed. In the final section of this chapter, important details of EARS are outlined.

2.1. Natural Language Processing

Machines and humans use languages to communicate. For a language, it is important to be expressive enough that all the things that are necessary things can be expressed through it. It is also important for a language to be exact enough. The producers and the interpreters of the language should be able to understand the message expressed through the language.

Natural languages (NL) have developed between natural entities, like humans. The topic of this thesis concerns only written NL, despite the existence of spoken and nonverbal NL. English was chosen as the only target NL.

The popularity in the Internet makes it possible to acquire various English language texts with relative ease. This is one of the main reasons English was chosen as the target language as the NL in this thesis. Because the texts in the Internet do not have to be professionally edited, however, they are usually not grammatically perfect. This has to be taken into account when using natural language processing (NLP) on texts from the internet. Humans tend to make grammar errors.

Languages that are not NLs also exist. They are typically made by humans for a certain purpose. For example, programming languages are languages, but not NLs, and controlled natural languages (CNLs) only contain parts of NLs. [2]

Humans learn to process natural language by getting the input through their senses and output through their bodies. Humans usually learn to handle language in various different forms in tandem – they get information through all their senses and can connect the information with words while they learn it – unlike a typical modern computer. In most cases, computer's only NL input is in text form. The computer does not automatically try to connect the text with any kind of meanings without special programming (machine learning) and other input sources (e.g. voice, images).

Natural language processing (NLP) is a field of computer science that studies ways to handle natural language (NL) through computer algorithms. NLP encompasses study areas from basic text processing to natural language understanding. In addition to text processing, modern NLP includes other forms of NLP as well (e.g. speech processing). [13]

This section has been divided into four parts. The first part consists of relevant information about natural language. The second part explains some natural language processing techniques. The third part describes background information about the information retrieval measures that are used in the testing part of this thesis. In the fourth part natural language understanding is discussed.

2.1.1. Natural Language

Typical humans use natural language (NL) every day. They communicate using audio signals and writing. A lot of humans even arrange their thoughts with the help of NL. Humans produce NL and invent new words and sayings; NL is in a constant state of change. NL can describe events and things that exist and even imaginary events and things. It may be impossible to record everything NL can describe without modelling all things that could be perceived or imagined by humans.

The theory of Universals of Grammar states that all natural languages contain some similar structures, they just need to be found if they have not been found. [14] These structures (like sentences, clauses, and phrases) are useful for identifying which parts of each utterance are connected to each other in which way. When constructing a controlled natural language (CNL) out of a NL, it is possible to only allow specific structures within the CNL.

English in text form was chosen as the NL to be processed in this thesis. To do so, basics of the English grammar must be known despite the fact that NL texts often do contain grammatical errors. This subsection consists of an overview of some very basics of the Universals of Grammar and English Grammar.

Universals of Grammar

The Universals of Grammar theory assumes that similarities between the grammars of different languages exist.[15] In 1963 Joseph Greenberg published a study in which he presented some linguistical universals concerning word order correlations [14], of which many have been later established as fitting in a larger number of languages than in the original study. [16] This indicates that there are similarities between different language grammars.

Searching for Universals of Grammar has aided the natural language processing (NLP) by finding different kinds of structures that appear in languages, even when some of them have been found to not appear in all languages. Automated analysis of languages can benefit from the research of different language structures.

Sentences consist of clauses and clauses consist of phrases. For example, the previous sentence consists of two clauses connected by "and". The beginning word "sentences" can be thought of as a noun phrase (NP) by itself and "consist of clauses" as a verb phrase (VP). A verb (e.g. be, use, calculate) is an action. A noun (e.g. computer, programmer, idea) is a thing. [17]

One of the findings in search for Universals of Grammar has been that most languages contain subjects, objects and verbs. Also, different languages can have different dominant orders of them. Subject is a word or a phrase that describes the actor or actors of the action, verb. Object is a word or a phrase that is the target for the action. For example, in English, the dominant order is "SVO", subject-verb-object (see Figure 2). [14]

Matthew Dryer presented the Branching Direction Theory (BDT) in 1992 [16] and some improvements to it in 2009 [18]. BDT states that languages can have right- and left-branching structures and usually one of them is more common. The sentences can be arranged in sentence tree structures (e.g. Figure 1) and the branching direction refers to which side of the tree has larger (deeper) branches. The sentence tree structure is used a lot in natural language processing (NLP). With it, the information of the sentence is stored in a human and machine readable format.

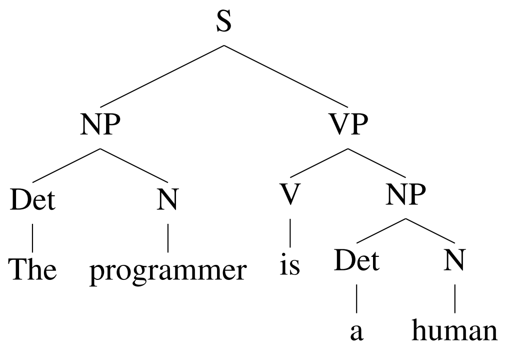


Figure 1. The example sentence, "The programmer is a human", in tree form. As can be seen, the structure's branching leans on the right. However, it does have left-branching parts, like "The programmer", where the "programmer" is the main word and "The" is attached to it from the left side. The order is SVO, where the subject is "The programmer", the verb is "is", and the object is "a human".

English Grammar

English is a dominantly right-branching language. [18] It also contains lots of helper words to show the relationships between words in sentences, which makes it an analytic language. Like most other languages, English contains noun phrases (NP) and verb phrases (VP), expressions of who did what. [17]

In English, phrases can be classified in three universal categories: subject, object, and verb. In greenbergian terms [14] their dominant order is SVO (Figure 2), subject-verb-object, and it can also be OVS when in passive form (Figure 3).

$$\begin{tikzcd} Subject \arrow[r] & Predicate \arrow[r] & Object \end{tikzcd}$$

$$\begin{tikzcd} The\ \ programmer \arrow[r] & uses \arrow[r] & the\ \ computer \end{tikzcd}$$

Figure 2. Typical active English sentence order. "The programmer" is the subject, "uses" is the verb and "the computer" is the object.

$$\begin{tikzcd} Object \arrow[r] & Predicate(\arrow[r] & Subject) \end{tikzcd}$$

$$\begin{tikzcd} The\ \ computer \arrow[r] & is\ \ used\ \ (\arrow[r] & by\ \ the\ \ programmer) \end{tikzcd}$$

Figure 3. Typical passive English sentence order. In passive sentences the subject might not be declared at all."The computer" is the object, "is used" is the predicate and "the programmer" is the subject, which is given to the sentence with the word "by".

Controlled Natural Language

A controlled natural language (CNL) is a language that consists of parts of natural language (NL) that are used to build a restricted set of rules or words. Controlled natural languages are created for specific purposes, like writing technical documents or more effective communication through a simplified language. The exact definition of a CNL tends to vary from author to author and the definition used in this thesis complies with the definition in the study by Kuhn (2014) [2].

Different CNLs vary in the language they are based on (although English is the most common base language), the level of restrictiveness (some contain only a small vocabulary and few possible sentence structures, when others only omit small parts of NL) and the environment of origin (academia, industry, government).[2] For example, Easy Approach to Requirements Syntax (EARS) [19] uses English as its base language. EARS applies templates in which NL is fitted in. EARS originates from industry, as it is meant for requirements documents.

Some CNLs can be straightforwardly mapped into a formal form, like first-order logic, without eliminating the readability by humans. For example, Attempto Controlled English (ACE) [20] can be turned into first-order logic and it is easy to read by humans, but writing it requires deep knowledge about how the denotations work within each sentence's context. Because of this, it is not easy to automatically translate NL into ACE, or another similar CNL.

In contrast to ACE, turning NL into EARS may not require quite as rigorous understanding of the NL, although it would help. EARS consists of the patterns and pieces of NL within the patterns. When transforming NL into EARS, everything may not have to be understood by the translator to create properly structured EARS sentences.

2.1.2. Natural Language Processing Techniques

Natural language can be processed algorithmically with computers. Various NLP methods to do so vary from very basic level processing, like separating words from each other, to more advanced and complicated ones, like information extraction, machine translation and opinion mining. [13] The more complicated and advanced methods usually build on the very basic NLP techniques.

An SRS usually consists mostly of text. Thus, this thesis is focused on the written text processing side of NLP. In this subsection is presented some of the most relevant NLP techniques.

Tokenising

Tokenising is separating things, like sentences or words, from text. The tokenising is done for better analysis of texts. Separating sentences from each other helps with handling the text as units. Word tokens can, for example, be compared against each other and against different lexicons and they can be modified or connected with interpretations.

For most words, tokenising means simply picking the groups of characters between spaces or other invisible characters. However, special characters and punctuation marks are sometimes used as parts of words, sometimes to mark an abbreviation and sometimes to mark the end of a sentence. Thus, tokenising words can require a good knowledge base of the practices used for mark-up. Also some machine learning might be required, since all word-mark-up combinations may not have been documented.

Tagging

Tagging can be used to tag tokens and phrases with labels that tell the purpose of the token or phrase. For example, POS-tagging tags the tokens with their part-of-speech (POS) labels and NER-tagging tags named entities, like location names. The software implementation presented in this thesis relies on POS-tagging.

Stanford POS-tagger uses Penn Treebank tag set [21] for English language tagging. It uses the maximum entropy model to calculate likelihoods for different tags and the likelihoods are based on the training data. It is one of the most popular and distinguished POS-taggers available. [22, 23, 24] Thus, it was chosen as the POS-tagger for the EARS Converter [1].

Named entity recognition (NER) makes it easier to POS-tag words more correctly. Named entities often contain words that either have other meanings or have no known meanings as normal (not named entity) words. For example, "University of Oulu" is a named entity which further contains a typical word "University" which could refer to any kind of university, and the named entity, location name, "Oulu", which is hard to interpret as anything else than as the name of the city. However, NER is often not so

[1]Chapter 3 Sections 3.3-3.5.

straightforward and to have NER for SRS, lots of knowledge of the language used on the specific SRS would be required. [25]

Parsing and Parse Trees

Parsing in NLP refers to the act of forming tree structures out of sentences. The tree structures, parse trees, describe how parts of sentences are related to other parts of sentences. For example, many parsers sort POS-tagged words into trees that describe phrase structures (e.g. Figure 1). Parsers facilitate the automated analysis of sentence structures.

Parse trees contain syntax of strings. They should not be confused with semantic trees that are used for finding the meanings rather than the syntax. [26, 27] But semantic trees, among other lexical tools, can be used to find the syntactic connections between words. That way they can be used to build parse trees.

Language parse trees and pattern recognition trees have already been combined to make computers produce descriptions of pictures. Language structures are hierarchial, like human perception of what a picture contains. For example, a picture might contain grass, part of a building and a part of a human and that could be interpreted as "a human stands in front of a house", when the human is interpreted as the main object of the picture, the building as a secondary object and the grass as irrelevant background. [28]

Natural Language Toolkit

Natural Language Toolkit (NLTK) is a Python library of libraries that contain different tools to use in natural language processing (NLP). The version used in this thesis is 3.3. NLTK can be used for producing different parse tree formats, tokenising sentences and tagging words. NLTK also contains other useful functions, like visualisation tools for the parse trees. [29, 30]

Keyword Extraction

Extracting the keywords of a text assists automatic recognition of the text's topic. It can be handy in information extraction of large masses of data when only a general idea of what is written about is required. [31]

Extracting keywords usually involves removing stop words. Stop words are words that have little or no meaning. English stop words usually include a lot of helper words, like "and", "the", "if", "when", and so on. The stop words may not have much meaning by themselves, but they are important for the sentence structure when trying to decipher a single sentence rather than a general idea about the text as a whole.

Keyword extraction is not very useful in the basic level of transforming NL into Easy Approach to Requirements Syntax (EARS) form, since the structure is more important than topic in that level. Thus, it was not included in the current version of the EARS Converter (presented in Implementation chapter). Nonetheless, keyword extraction could be useful in further development of translating texts. When trying to rebuild sentences that are so ambiguous that even the structure is not recognised properly, keyword extraction can give extra information about the context.

2.1.3. Information Extraction Measures

Information extraction systems resemble classification systems in the sense that they both classify data. Information extraction from text data includes classifying the texts and parts of texts as relevant, irrelevant, correct and incorrect among other more specific possible classification. [32] For example, in keyword extraction, parts of texts are classified as possibly containing keywords, and the words in those parts are classified as keywords and non-keywords.

Converting natural language sentences into a controlled natural language sentence format is an information extraction system in the sense that it attempts to extract the relevant information from NL sentences to create their CNL equivalents. To measure the effectiveness of the system, some typical information retrieval measures can be used on it. In this thesis precision, recall and F-measure are applied for that purpose.

Table 1 depicts the areas that are measured with precision, recall and F-measure. In the *Relevant* column are the true positives (TP), which are information that gets extracted and is relevant, and the false negatives (FN), which are non-extracted information which is relevant. In the *Irrelevant* column are the false positives (FP) and the true negatives (TN).

Table 1. Information extraction success for precision, recall and F-measure

	Relevant	Irrelevant
Extracted	TP	FP
Abandoned	FN	TN

Precision

In data classification, precision tells how many of the positive results were actually positive. In information extraction, precision tells how much of the extracted information is also relevant information. [32]

In Equation 1 is shown how precision is calculated. *P* is precision, *TP* the number of true positives and *FP* the number of false positives. In contrast, Equation 2 shows how precision calculated using information extraction terminology. *P* is still precision, *relevant* refers to the group of relevant information and *extracted* to the group of all of the extracted information. [32]

$$P = \frac{TP}{TP + FP} \tag{1}$$

$$P = \frac{relevant \cap extracted}{extracted} \tag{2}$$

A problem with precision is that it is only as good as the analysis of how well the relevance of the information is assessed. This can make it highly biased. Also, it is a binary measure in the sense that the extracted units need to be classified in binary categories. The binarity does not take into account that some relevant data may be more relevant than the other. Because of this, when precision is used as an information extraction measure, the assessment of relevance must be well recorded. If the relevance is not binary, it might be sensible to consider how to take into account the strength of relevance. [33]

Recall

Recall is the measure of how much of the correct information was classified as correct. In information extraction, recall describes how much of the relevant information was extracted. Unlike precision, calculating recall requires knowledge of how much relevant information exists within the test data. [32]

In Equation 3 is shown how recall (*R*) is calculated in typical classification systems. *TP* is the number of true positives and *FN* is the number of false negatives. Equation 4 shows how recall is calculated using information extraction terminology. *R* is still recall, *relevant* refers to the group of relevant information and *extracted* to the group of all of the extracted information. [32]

$$R = \frac{TP}{TP + FN} \tag{3}$$

$$R = \frac{relevant \cap extracted}{relevant} \tag{4}$$

Like with precision, also with recall the accuracy how relevance is assessed is not taken into account within the measure. With recall, special attention should be paid to how the amount of actual relevant information within the dataset is calculated.[33]

F-Measure

F-measure (also known as F-score) is a popular measure in information retrieval systems, and it can also be applied to information extraction systems. It combines precision and recall to measure the overall success of information retrieval, or extraction, by calculating their harmonic average. However, like precision and recall, its accuracy depends on the accuracy of the measuring of relevance. [32, 33]

Equation 5 shows how the F-measure (*F*) is calculated. *P* represents precision and *R* recall. [32]

$$F = 2 * \frac{P * R}{P + R} \tag{5}$$

2.1.4. Natural Language Understanding

Natural language understanding (NLU) is the field of study in machine learning that focuses on making the machines understand natural language (NL) in one way or another. An artificial intelligence (AI) may have to have at least human level understanding to comprehend the meaning of NL. In other words, it has been claimed that NLU is an AI-complete problem.[34]

NLU encompasses various fields. Examples of them include sentiment analysis (automatic understanding of the tone of a text, is it positive or negative or something else)[35, 36], biomedical text mining and universal dependencies (how words are linked to each other within texts)[11], and interpreting NL in logic form [37]. To build a system that would understand NL on human level, many areas of NLU might have to be combined.

NLU deals with topics somewhat obscure topics, like language ambiguity and complexity and machine reading comprehension (Section 2.2). Human understanding of language is intrinsic, which can make it difficult for humans to reverse engineer their natural language understanding. This thesis is focused on attempting to reduce the ambiguity of NL so that it would become easier to navigate through the obscurity and find out how to make machines understand NL.

2.2. Ambiguity and Complexity of Natural Language

For SRS it is important that they convey the information exactly the way it is meant to be. The ambiguity of natural language (NL) hinders the precision of expressing the requirements. If the requirements are written ambiguously, the developers can not be sure whether or not they created a product that complies with the customer's will. If the product is not what the customer wanted or needed, it can cause great costs and delays.

On the other hand, the reading comprehension can suffer when the features are written meticulously without paying attention to the readability. The text structures can become too complicated to be processed easily by humans. Both, the ambiguity and the complexity of NL, make the reading comprehension of SRS documents more challenging.

This section first tries to disambiguate what ambiguity is and then describes some relevant ambiguity measures. In the third subsection complexity is discussed in more detail, as it is tightly related to ambiguity within NL context. In last subsection some points about reading comprehension are brought up.

2.2.1. Ambiguity

Even the definition of ambiguity is ambiguous. [38, 39] However, the ambiguity of NL enables its versatile usage. It has been noted, that new words are allowed to have several possible meanings when introduced to a language before some of them are established, which indicates that the ambiguity of NL is not decreasing. [40] To face the challenge of the ambiguity of NL, finding ways to disambiguate is important.

Humans can disambiguate NL by deciphering it through the context it appears in. They naturally interpret ambiguous language as unambiguous, unless they find a reason to suspect there might be more than one way to interpret something. Detecting ambiguity is seldom easy. [38]

Resolving ambiguity automatically has been attempted. [41, 1] But even with translating language it is difficult.[42] In this thesis some of the ambiguity of NL in SRS is resolved by transforming the sentence structure into a less ambiguous one, which faces some of the same difficulties machine translation does with ambiguity. For example, the words may not appear in the dictionary used and despite correct placement of words, the meaning may still stay ambiguous because of the lexical ambiguity of the words.

Many types of ambiguity exist for NL. Different texts discussing ambiguity have different, often ambiguous, ways of categorising different types of ambiguities in groups.[40, 42, 4, 43] Despite that, ambiguity can be categorised. Some of the most relevant types of ambiguity in the context of this thesis are semantic, structural and scopal ambiguity.

Semantic ambiguity

Semantic ambiguity is the type of ambiguity that deals with the meaning, the possible interpretations, of text. Words can have multiple meanings, phrases and clauses and sentences can have multiple meanings, sentence groups can have multiple different meanings, and so on. The very base of semantic ambiguity of texts is lexical ambiguity, which can be measured by the number of meanings a word can have.

If a word has more than one meaning, it is lexically ambiguous. A couple of examples of lexically ambiguous words include "bank" (e.g. the place money is stored in or a river bank), "die" (e.g. singular of dice or the opposite of live), and "fork" (e.g. the kitchen utensil or branch). It is easy to prove a word has multiple meanings by just looking at a dictionary. However, it can be harder to prove that it does not have more than one meaning, since language is constantly evolving and the meanings of words also depend on the context.

Structural ambiguity

Structural ambiguity is about sentence structure. Sentence structure can restrict the possible interpretations of a word depending on which word follows which and what kind of punctuation is used. If the structure is ambiguous, it can be difficult to determine what words the other words are referring to.

If a word appears in one kind of structure and then in another, it can have different meanings. For example, "this is a sentence" and "is this a sentence" do not have the same meaning despite consisting of the same words. Because of this, structural ambiguity and lexical ambiguity are related to each other. In addition to that, the more there are words in a sentence, the more complicated the structure can be. If a structure is complicated, it has more potential for more interpretations than a simple structure.

Scopal ambiguity

Scopal ambiguity relates to the scope words refer to. It can be the scope of an adjective (e.g. in "the new program part" the scope of "new" can encompass just "program" or "program part") or a pronoun (e.g. "it" can refer to many different things) or a quantifier (e.g. "a program should run every second" can refer to a certain program that should be run multiple times or that some program, no matter which one, should be run).

2.2.2. Ambiguity Measures

Measuring ambiguity can facilitate writing less ambiguous requirements. If a sentence in SRS is deemed ambiguous before the SRS is released for use, it can be fixed before it causes trouble. To some extent, ambiguity can be measured in semantic and syntactic level. [44]

Kiyavitskaya et al. [4] presented requirements for systems identifying ambiguities in NL requirements. The requirements include two main parts: 1. The system must identify which sentences are ambiguous and 2. for each identified ambiguous sentence the system must provide help for understanding why the sentence is ambiguous.

It is challenging to create objective measures of ambiguity when a message's ambiguity depends on the subjective interpretations made by the sender and the receiver of the message. Measuring ambiguity can be done with indirect ways if the ambiguity type is well defined.

Lexical ambiguity measures are one of the most common ways to measure ambiguity in requirements engineering. Measuring lexical ambiguity is possible to do by determining how many different definitions a word has. One way to do so is to look in a dictionary and see how many definitions there are for the word. However, a dictionary may not contain all of the possible definitions.

Different software projects have differently named elements. A common dictionary that encompasses all possible words in software requirements does not exist. For specific SRS documents, however, it is technically possible to build a specific dictionary for the words. That may not be feasible for SRS documents in general, just for specific SRS documents.

2.2.3. Complexity

When talking about natural language, complexity refers to the complicated structures of multiple words and multiple combinations of words. Information complexity refers to the multiform ways information is arranged. Basically, language complexity is information complexity, but information complexity is not necessarily language complexity. Several definitions for language complexity exist, but for the purposes of this study, the meaning as the structural complexity of language [45] is the most interesting one.

Ambiguity is not complexity and complexity is not necessarily ambiguity. A sentence structure can be very complicated, but it still can have only one meaning. Sentence structure can also be very simple, but it still can have many meanings because of semantic ambiguity. However, if a sentence structure is complex, it is more difficult to interpret than when it is simple. With complex structures there are more pieces that need to be comprehended together.

Information content

Information content can be complex, it can have multiple parts that relate to each other in different ways. The parts and the ways they relate to each other are both information content. The complexity of information content can be measured from the numbers of different characters and other structures. [46]

Kolmogorov complexity means the shortest form a text can be written in without losing its information content. Nowadays it is often interpreted as the shortest program that can output the text. In Figure 4 is presented one way how words can be encoded to see their Kolmogorov complexity in an encoding style.[46]

$$\begin{tikzcd}
10 & 10 & 10 \\
Complexity \arrow[d] & heavyheavy \arrow[d] & nooooooooo \arrow[d] \\
Complexity & 2 * heavy & n + 9 * o \\
10 & 7 & 5
\end{tikzcd}$$

Figure 4. An example of Kolmogorov complexity of encoded words. Three pieces of text that have the same length can have different Kolmogorov complexities: For "Complexity" it is 10, for "heavyheavy" 7, and for "nooooooooo" it is 5.

Kolmogorov complexity tells the amount of information content in a string. It can also be used as an indirect complexity measure. [46] To calculate the Kolmogorov complexity of a text file, a file compressor can be used. The smaller the packed file, the lesser the information content. For testing the datasets used in the implementation of this thesis, 7zip was used as the file compressor. [47]

Software Complexity Measures

In the field of Software Engineering, complexity generally refers to the system and how its parts interact with each other. The term complicatedness is closer to the linguistic interpretation of complexity, where complexity refers to the language rather than the system it describes.

Lines of Code (LoC) is one of the easiest and most common ways to measure software complexity. [48] Calculating the number of sentences in a NL is somewhat similar to the LoC measure. Each sentence represents a piece of information, a bit like lines of code do.

Language Complexity Measures

Language complexity (and ambiguity) has been an object of interest in linguistics especially when talking about cross-linguistic studies. Results of comparing the complexities between different languages have even been thought of as evidence of different levels of intelligence between cultures. However, such ideas have been deemed as political rather than scientific in modern studies. [49] A more practical use of linguistic complexity is using complexity as a measure for how difficult it is to understand a text.

An expert linguist might be able to measure morphological complexity, e.g., by calculating different morphemes and their relationships within the text. [45] However, creating an automated system for that purpose is not simple. When creating a system to automatically analyse complexity, it can be easier to concentrate on the complexity of syntactic structures.Syntactic complexity measures include lengths of phrases, number phrases in clause, number of clauses in unit and number of word order patterns. [45]

Noun Phrase Chunks

Chao Y. Din presented the use of noun phrase chunks (NPC) as a complexity and quality measure specifically for requirements documents.[50] Din proposes three ways to measure the complexity using NPCs: NPC-Count, NPC-Cohesion and NPC-Coupling.

NPCs describe the sentence structure by separating the noun phrases (NP) from the sentence structure. A NPC consists of the main noun and words that specify and describe it. A few examples of NPCs include "The programmer", "a programming programmer" and "that new skilled programmer", where "programmer" is the main noun. A NPC can be treated as a single noun. For example, in the sentences "NPC programs" and "there was NPC", "NPC" can be replaced by any of the previous NPCs and the sentences make sense.

The NPC-Count measure calculates the number of NPCs in a sentence. It describes the number of different entities that the sentence is about. The more there are NPCs in a sentence, the more complex it is, since its meaning can consist of at least that many parts.

The NPC-Cohesion is measured by calculating the sum of all sentence cluster sizes within a requirement and dividing it by the number of all sentences in the requirement. A cluster of sentences is formed so that a sentence and the next one belong to the same cluster if both sentences contain the same NPC. [50]

NPC-Coupling is measured by calculating the sum of spatial distances between NPCs that are inside and outside a sentence within the target text. [50] Measuring the Coupling this way requires defining spatial distances between NPCs and sentences. Measuring Coupling could also be useful in analysing how interrelated the requirements are, but that is out of scope of this thesis and Coupling is not measured for the current implementation.

2.2.4. Reading Comprehension

Reading comprehension is a topic of importance in both linguistics and NLP. Reading comprehension is what natural language understanding (NLU) attempts to reach with machines. Reading comprehension is more difficult on ambiguous texts, but at least human readers can fill in the holes and comprehend texts at times even when they are ambiguous, if they know what to expect from the text.[51]

Humans can process only a small number of units at a time [52] when machines can handle as many as they are built to handle at a time. This makes it possible for a machine to go through complex structures without making errors more easily, assuming there are no errors in the structures.

For example, the sentence "the programmer programs" can be handled as three units, three words, by a human, but a computer can remember all the 23 characters at the same time. When human processes the sentence, they can still divide each unit into smaller units to reproduce the text, but if the units are interpreted incorrectly, they might produce an incorrect text, for example by forgetting the article as meaningless ("programmer programs") or by recalling the spelling incorrectly ("the programer programs"). In contrast, a machine could store the whole sentence in memory without changing its format and it could still create many different kinds of units (e.g. words, parts of words) in which it could classify the sentence parts.

Humans vary in reading comprehension abilities depending on their background and the target reading material. [53] A lot of studies have been made on school children to develop better teaching methods. Adults have not been studied as much, but they still also vary in reading comprehension abilities. [54] The equivalent of reading comprehension variation in machines is the training datasets and the algorithms used to interpret the text. Vocabulary size has a great effect on reading comprehension. [55, 56]

To some extent the readability of a text can be measured. One of those measures is the Automated Readability Index (ARI), which uses the stroke count (number of non-whitespace characters), word count and sentence count. How ARI is calculated can be seen in Equation 6. The Grade Level (GL) corresponds to the US school system grades levels. [57] A problem with ARI is that it does not take the vocabulary or structure into account at all, it only measures superficial readability.

$$GL = 4{,}71(\frac{strokes}{words}) + 0{,}5(\frac{words}{sentences}) - 21{,}43 \tag{6}$$

2.3. Software Requirements Specification

Requirements engineering (RE) consists of requirement elicitation, analysis and negotiation, documentation, validation, and management. [58] The document containing software requirements is called the Software Requirements Specification (SRS). In the SRS the elicitation of requirements and their analysis and negotiation are culminated in the documentation of the requirements.

In requirements documentation the SRS is created so that the supplier and the client are in agreement about the requirements and can verify them. To define software requirements, discussions between the customer and the the software provider need to take place. Those discussions are typically held in natural language (NL) whether they are done in speech or writing or both. Since the discussions are in NL, it is straightforward to write the requirements in NL.

The field of software engineering consists of areas like information engineering and embedded engineering. Especially in embedded software engineering, where human lives can be at stake (pace makers, car software, etc.), software quality is important. If you release a faulty software, the cost of patching it up is much larger with embedded software. One could argue that the better the SRS, the better the quality of the software, since the better you understand what is required, the more accurately you can create it. [59]

SRS documents are used to define what is expected from the software product. They can be used by those who implement the code and those who validate it. A big part of the issues in implementing software lies in poor SRS documentation and it causes costs and delay. Unfortunately, it may be impossible to predict all the possible changes in requirements, but a thorough groundwork can save a lot of time and money.[60, 59]

This section consists of three subsections. The first subsection describes what is meant by SRS quality, the second discusses the language used in SRS and the third the structures of SRS.

2.3.1. SRS Quality

It could be said that the quality of SRS documents could be measured by the number of errors in them. There are two kinds of errors that are made during writing SRS documents: knowledge errors and specification errors. Knowledge errors are not always preventable, since the required knowledge to do so may be available only later in the process. However, specification errors should always be preventable by writing the SRS better. [61]

Davis et al (1993) found 24 specific qualities an SRS should exhibit for being a good quality SRS [61]. Some of them overlap. For example, unambiguous and understandable come close to being synonyms when considering from human perspective and modifiable could be thought to include concise and reusable. Also, quality measures may contradict each other and it is impossible to make a perfect SRS in general sense. [61]

The following subsections discuss the most relevant ones of the 24 SRS quality measures of [61] and how they are related to this thesis.

Unambiguous

In SRS documents ambiguity can lead to mixed interpretations of the requirements' meanings. If the ambiguities are not fixed early on, the developers may produce software which lacks intended features and exhibits unintended ones. The testers may still catch the incorrect features, but it costs a lot more than catching them while defining the desired features. If the faulty software is handed over to the customer, the costs of correcting the errors caused by ambiguous requirements are even larger.

A sentence's ambiguity can be measured by how many different interpretations the sentence has. The sentence is unambiguous if it only has one interpretation. Natural language sentences can be interpreted in various ways depending on the context. With natural language, one sentence can have so many meanings that it could be said that natural language is inherently ambiguous.

Using formal form for natural language sentences can help with reducing ambiguity, despite the possibility of it also reducing understandability. [61] Formal form can be understood as a strict context in which the language may be used only in predefined ways and so the interpretations are limited.

SRS ambiguity can be sorted in two categories: linguistic ambiguity and software engineering specific ambiguity. [1] Linguistic ambiguity can be resolved by a person who is not familiar with the software described in the SRS as long as the person has relatively good language skills. It is related to grammar, syntax, and how the sentences have otherwise been built. For resolving software engineering related ambiguities an expert on the system is required. The software engineering related ambiguities include things like conflicting requirements and not describing a requirement in the required level of precision.

There are systems that have been built for detecting ambiguity in natural language SRS documents.[1] However, no gold standard ambiguity detecting tool of such kind exists yet. Even manually detecting ambiguities by reviewing the SRS is not always reliable due to human errors.

Ambiguity in technical documents, especially in SRS documents, leads to delays and errors due to misunderstandings. It may be useful to be able to produce at least some level of code straight from its technical specifications, but if the specifications are ambiguous, that is very difficult, if not impossible, without someone to disambiguate them first.[62]

Using specific requirements management tools helps with removing some of the ambiguity by automating the structure creation. Using CNLs or formal languages lessen the ambiguity, by removing the ambiguities of NL.

Understandable

SRS's understandability means how easily the readers of the SRS interpret correctly what is said in it. As it sounds, it is not easy to measure accurately. In some cases, the developers and testers might understand UML charts and other formal representations of the requirements better than natural language requirements, but the other stakeholders, like customers, need to be able to understand the requirements easily as well.

Understandability is closely related to the unambiguousness of SRS. However, understandability takes into account how humans see the documents. A sentence can be unambiguous, but it can also be hard to comprehend if it is too complicated and long. Understandability can be enhanced by keeping things simple and clear.

Modifiable

The SRS should be modifiable so that changing it is trivially easy, as the requirements change when new information comes in. Electronically stored SRS documents can be easily modifiable depending on the form they are stored in. Typically, to modify the text, you only need a text processor with electronic documents. The base level of modifiability is that the SRS can be modified.

Tracking the requirements and the changes done in them can increase modifiability by helping to revert wrong changes and knowing which parts to modify. Making a change in one place of the SRS may require making changes in some other places too when the requirements depend on each other.

Keeping the requirements modular can also increase modifiability. It can also make easier to track them. If the requirements have names and clear borders, they may be thought of as modular. Naming the requirements also makes it easier to arrange them and increases their modularity.

Executable/Interpretable

Usually SRS documents are not executable or interpretable in the sense that it would be possible to press a button and get code generated out of the SRS. For that to be possible, there would have to either be an AI capable of understanding NL or the SRS would have to be written in an executable language. [61]

Reusable

The requirements in the SRS can be reusable in other SRS documents if they are easy to modify to reflect the needs of the other SRS documents or if they are re-applicable without modification. Like in software, where modularity is one dimension that can make the code more reusable, the SRS is also more reusable if it is modular. A good modular SRS has separate requirements that are stated clearly and on the right level of generality. [61]

2.3.2. Language

SRS documents are typically written in English, which is a natural language (NL). Other NLs can also be used. Many SRS are also written in a controlled natural language (CNL) or a formal language, like mathematical equations. Pictures can also be used in SRS, for example, in the form of UML diagrams. [63, 12]

SRS documents are usually not made public, so it is hard to find verified facts about the quality of language used in them. However, as the writers of SRS have been mostly human to this day, the documents are bound to have errors. That is why the SRS needs to be under quality assurance. Unfortunately, the quality of grammar and spelling in the SRS are not the top priority, so low quality language is to be expected in the documents.

Natural Language

NL is so often used in SRS because of its universality. Even people who have not been taught technical CNLs can easily write and read text in NL. Natural language can be used to express a wider variety of things than any CNL. Compared to picture data, NL text data can also be cheaper to store in memory.

Unfortunately, NL in an SRS is only as precise and grammatically correct as its writers write it. Humans make a lot of spelling errors, of which many can be detected and corrected with the help of current technology. [64] Since SRS writers do not always use spelling checkers and even if they do, they can still make mistakes. Many SRS documents contain grammatical and spelling errors in addition to ambiguity that does not stem from wrong usage of language.

Controlled Natural Language

Technical documents that are not written in NL are often written in a CNL. A CNL can greatly clarify the meaning of sentence, which greatly reduces the ambiguity NL can possess.

For the purposes of this thesis, Easy Approach to Requirements Syntax (EARS)[2] was chosen as the target CNL. Unlike, for example Attempto Controlled English (ACE), which can have unlimited combinations of language structures, EARS relies on a limited number of different patterns.

Unified Modeling Language

Unified Modeling Language (UML) depicts complex algorithms and systems as pictures. It can be used for software development, and in SRS documents, to describe things that are difficult to do in natural language. Despite its allure, this thesis focuses only on NL instead of UML.

2.3.3. Structure

The SRS consists of specifications of requirements for the desired system. If the SRS is in the form of a single document, it can have an introduction that explains what the SRS is for and what abbreviations are used. It can also include, for example, descriptions of system parts. If the SRS is stored in a requirements management system (RMS), it can still contain the descriptions, but they are not arranged in the single document format. However, RMS may be able to produce a single SRS document, if it is necessary.

[2]There is a whole section dedicated to EARS in this chapter (Section 2.4).

Single document

The SRS can be in a single document format. It can be written in .pdf, .doc, .txt or practically any other document type. If the SRS contains pictures or diagrams, most likely it is not in a raw text format, but in a mixed media format. It can also be a physical document instead of being stored electronically.

If SRS is stored in a single document, it can be made to follow the IEEE Standards for Requirements Specifications. [65, 66]

Requirement management systems

In requirement management systems it makes sense to store the SRS as separate requirements specifications instead of keeping them all in one document. That way they can be connected through the requirement management system in ways that are hard to implement for s single document format.

2.4. Easy Approach to Requirements Syntax

The Easy Approach to Requirements Syntax (EARS) is a methodological way of writing SRS documents by using five simple templates. Requirements documents written using EARS have been proven to be less ambiguous than documents written in unrestricted natural language (NL). [67]

EARS was created at Rolls-Royce around 2009. [10, 19] Its initial purpose was to create better requirement documents for their products, including aero engine control systems. In other words, EARS was created for safety, since having better requirements leads to more reliability in designing the systems. It is easier to detect faults and fix them in requirements that are written in a way that is clear and easy to understand.

For a controlled natural language (CNL), EARS is relatively simple and easy to learn. EARS does not limit the requirements in strict ontologies, except for the templates, patterns, it uses. The EARS methodology consists of five patterns: Ubiquitous, Event-Driven, State-Driven, Optional Feature and Unwanted Behaviour. To an extent, they can be combined to make a sixth pattern group, the Combination.

The EARS patterns consist of a condition, an actor and an action, except Ubiquitous pattern that consists of only the actor and the action. In Table 2, each basic pattern is listed with its name, structure and an example. The actor is denoted with "the" and the action with "shall". Each of the four condition types have a different word it begins with (when, while, where, if) and the Unwanted Behaviour pattern's condition has also "then" at its end.

Table 2. A summary of the EARS patterns

Name	Structure	Example
Ubiquitous	**The** [ ] **shall** [ ].	**The** first protocol **shall** be implemented.
Event-Driven	**When** [ ], **the** [ ] **shall** [ ].	**When** the authenticate command is received, **the** first protocol **shall** activate.
State-Driven	**While** [ ], **the** [ ] **shall** [ ].	**While** the first protocol is active, **the** second protocol **shall** encrypt all messages.
Optional Feature	**Where** [ ], **the** [ ] **shall** [ ].	**Where** encryption is defined, **the** second protocol **shall** use the defined encryption.
Unwanted Behaviour	**If** [ ], **then** **the** [ ] **shall** [ ].	**If** a received key is wrong, **then the** first protocol **shall** terminate.

The work on EARS-CTRL has proven that EARS can be used to further translating natural language straight into code. EARS-CTRL is a system that turns EARS text into a linear temporal logic (LTL) format, which in turn can be turned into C code with the help of Simulink. The EARS-CTRL demands a glossary of the components so that they can be referred to with (restricted, EARS form) natural language. [68, 69]

2.4.1. The Ubiquitous Pattern

The Ubiquitous Patterns (UP) describe an inherent part of the systems that is not a response to an action or otherwise optional. For example, it can describe what sort of parts the software needs to include and what function which part performs.

The Figure 5 shows the structure of the pattern. UP consists of the main parts of a requirement the *actor* and the *action*. The actor is expressed starting with "the" and the action with "shall". If the action is in passive form, the "shall" is combined with "be". It is possible to forbid the use of such passive form, but it is allowed in the context of this thesis.

$$\boxed{The\underline{\quad(1)\quad}\qquad shall\underline{\quad(2)\quad}}$$

Figure 5. The Ubiquitous pattern. "The" and "shall" are marker words in the structure. (1) represents the named target system and together with "The" it forms the actor, (2) with "shall" represents the action it is supposed to do. The described action the system is supposed to do comes without any pre-conditions in this pattern.

UP could be said to be the basic pattern, it can be found within all of the other patterns. What distinguishes UP from the other EARS patterns, is the lack of condition part. The action executed by the actor of the UP is ubiquitous, the actor should always do it, when present, whatever the situation.

2.4.2. The Event-Driven Pattern

Event-Driven pattern (EDP) describes a requirement of a response for an event that triggers a behaviour. For example, in the example sentence from Table 2, "When the authenticate command is received, the first protocol shall activate", the triggering event is receiving the "authenticate command" and the response is that the "first protocol" "activates". In Figure 6, the pattern is described in detail.

$$\boxed{When\underline{\quad(1)\quad},\qquad the\underline{\quad(2)\quad}\qquad shall\underline{\quad(3)\quad}}$$

Figure 6. The Event-Driven pattern. The pattern is much like the Ubiquitous pattern with a "When"-condition put in front of it. (1) represents a description of the triggering event and with "When" it is the EDP condition. It may consist of an initiating action and preconditions. (2) is the target system and (3) is the response.

The EDP condition can include preconditions for the trigger. For example, "when the authenticate command is received after the initiate command" includes the trigger, "the authenticate command is received" and a precondition for the trigger, "after the initiate command". It has not been strictly defined within EARS methodology how the preconditions should be separated from the actual trigger. Thus it is the requirement engineer's responsibility to write the condition clearly in NL so that there is no doubt about what the trigger is.

2.4.3. The State-Driven Pattern

The State-Driven pattern (SDP) describes an expected system behaviour while it is in the defined state. In the example sentence from Table 2, "While the first protocol is active, the second protocol shall encrypt all messages", the state condition is "While the first protocol is active" and the rest are the actor and the action, the system behaviour in the state. Figure 7 shows the SDP in detail.

$$\boxed{While\underline{\quad(1)\quad},\qquad the\underline{\quad(2)\quad}\qquad shall\underline{\quad(3)\quad}}$$

Figure 7. The State-Driven Pattern. The SDP is very similar to the EDP, but instead of "when" it has "while" at its beginning and (1) represents the state in which (2), the system, should respond in what way, or continuously do something, which is described in (3).

2.4.4. The Optional Feature Pattern

The Optional Feature pattern (OFP) has a condition that describes a certain situation where the behaviour happens. The condition is not a triggering event or a state, but, for example, if some other feature has been implemented. The Optional Feature condition depends on other features.

In Figure 8 is the basic structure of the OFP. The OFP condition begins with the word "where" and continues the same way as the other patterns that have conditions.

$$\boxed{Where\underline{\quad(1)\quad},\qquad the\underline{\quad(2)\quad}\qquad shall\underline{\quad(3)\quad}}$$

Figure 8. The Optional Feature pattern. The pattern resembles the two previous patterns described in Figure 6 and Figure 7. It also has a condition, but it starts with the marker word "where" and describes the optional feature in (1). The behaviour is described in (2) and (3) like in the two previous patterns.

2.4.5. The Unwanted Behaviour Pattern

The Unwanted Behaviour pattern (UBP) differs from the other patterns most in the sense that it describes behaviour in expected error situations. It is technically possible to use EDP instead of UBP, but the knowledge about whether the condition is desirable or not would disappear. Some EARS users may decide to only use EDP, but UBP is included in the implementation of this thesis, albeit shallowly.

The UBP structure is described in Figure 9. It consists of a "if"-"then the"-"shall"-structure. Other pattern conditions do not have a defined ending word, like UBP condition does.

$$\boxed{If\underline{\quad(1)\quad},\qquad then\qquad the\underline{\quad(2)\quad}\qquad shall\underline{\quad(3)\quad}}$$

Figure 9. The Unwanted Behaviour pattern. This pattern has four marker words: "If", "then", "the" and "shall". In (1) between "If" and "then" should be the description of the unwanted behaviour (the UBP condition consists of "if", (1) and "then"). In (2) is the system and in the place (3) is how the system should respond to the unwanted behaviour.

The Unwanted Behaviour condition is an event that comes from outside of the system. For example, a user may give the wrong PIN-code or the user may write a too long text in a text box. The response can be something like prompting a new PIN-code for three times before locking or not responding to text input after a limit of 300 characters.

2.4.6. Combination of Patterns

The patterns can be combined with each other if has been decided that combinations of patterns is allowed. Combining patterns allows for more complex conditions for behaviour. However, if applied carelessly, combining the patterns may make the requirement more difficult to comprehend. In Figure 10 is an example of how a combination of patterns can be structured.

$While\underline{\quad(1)\quad}, \quad when\underline{\quad(2)\quad}, \quad the\underline{\quad(3)\quad} \quad shall\underline{\quad(4)\quad}$

Figure 10. An example of a Combination of patterns, which combines the State-Driven and Event-Driven patterns. This combination consists of two conditions, the actor and the action.

Combinations of patterns can be either banned or allowed, and for the implementation of this thesis, they were allowed. The types of allowed combinations of patterns are described in Figure 11. There are 45 different combination patterns when only one of each type of conditions are allowed and the Unwanted Behaviour pattern can only be the last condition before the actor.

$condition1$,	$condition2$,	$actor$	$action$		
$condition1$,	$condition2$,	$condition3$,	$actor$	$action$	
$condition1$,	$condition2$,	$condition3$,	$condition4$,	$actor$	$action$

Figure 11. The only allowed combination patterns in the implementation of this thesis. Any Condition can be Event-Driven, State-Driven, Optional Feature or Unwanted Behaviour, but only the last one before the actor can be Unwanted Behaviour, because of the "if...then" structure. There are only three possible combinations, because it is assumed that if there are two conditions of the same type, they can be merged together with "and". For example, the conditions "when the program starts" and "when the cursor moves" can be merged to create "when the program starts and the cursor moves".

English, as a typical natural language, is so flexible that a sentence can have a main clause and multiple subordinate clauses or it could even consist of more than one main clauses. Sentences with multiple subordinate clauses are more difficult to express in EARS if combinations of patterns are not allowed. A subordinate clause can often be translated into a condition.

3. IMPLEMENTATION

In this thesis is presented a tool for reducing ambiguity in SRS documents. Converting NL requirements sentences into EARS reduces the structural ambiguity by presenting only the cores of the sentences.

The tool presented in this thesis is called EARS Converter and it fulfills the main goals of a tool for identifying and measuring ambiguity, as defined by Kiyavitskaya et al. (2008) [4]. It determines ambiguity based on whether the sentence conforms to EARS patterns and it proposes an EARS pattern if it finds a structure that could be turned into one and warns if it does not find any.

Semantic ambiguity is more difficult to detect when structural ambiguity is also present. Resolving semantic ambiguity requires deep knowledge of the vocabulary of the target texts. [44] The focus of the EARS Converter is only in the structural ambiguity of the sentences. This way it can be applied on SRS data that describe more varied kinds of systems, since a glossary of the words and their meanings is not required. If the EARS Converter is developed further in the future, ways to detect semantic ambiguity in some level may be added.

EARS is a method of writing requirements in a simple and clear manner. If natural language requirements are turned in their EARS form, they might become less ambiguous. [62] If it is difficult or impossible to turn a natural language sentence in EARS form, that might be a sign that it is too complicated. The EARS Converter attempts to turn natural language sentences in their EARS forms.

The EARS Converter is the main implementation of this thesis. It takes raw text SRS requirements as input and converts them in their EARS form. In Figure 12 is shown the basic idea of the EARS Converter. The Converter consists of preprocessing, tokenising, tagging and parsing, and mapping to EARS phrases.

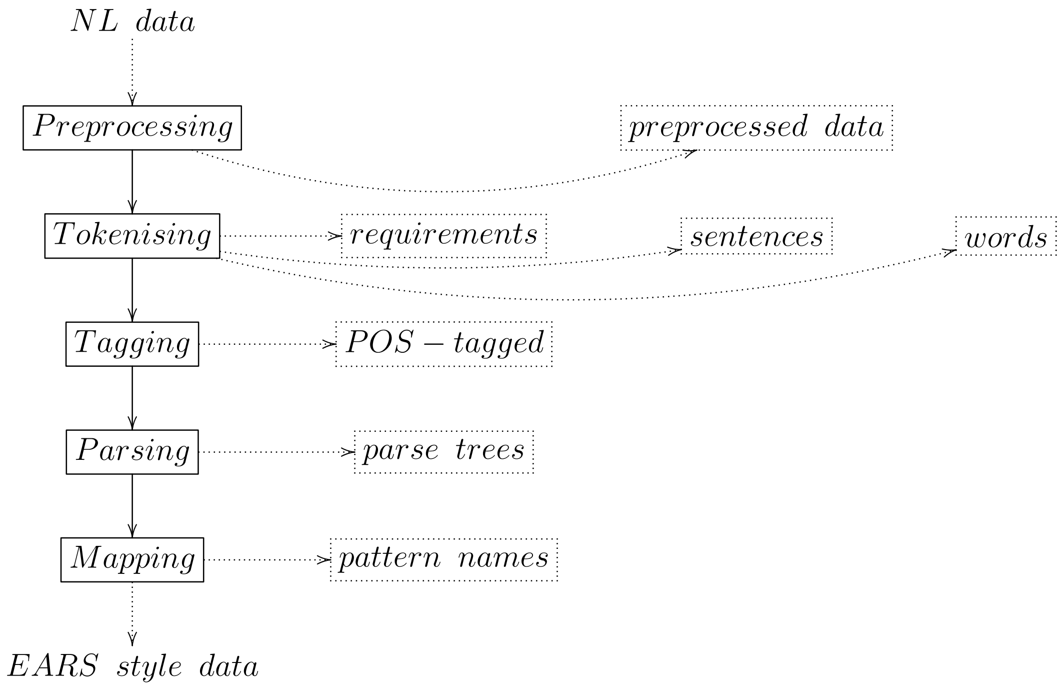


Figure 12. The EARS Converter's tasks, input, and output. The preprocessing task takes the input data and, from that, produces key words and preprocessed data. The tokenising task handles separating the preprocessed data into sentences and words. The tagging task labels the tokenised data with POS-tags. The parsing task creates parse trees out of the POS-tagged data. The mapping task turns parse trees into EARS sentences.

Algorithm 1 summarises the EARS Converter. Compared to Figure 12, tagging has been omitted, because in the current version of the Converter the parsing task handles both tagging and parsing.

Algorithm 1 The EARS Converter

```
Input: NL requirements specifications
Output: Requirements in EARS form
1: preprocessedData = Preprocessing(NLData)
2: tokenised = Tokenising(preprocessed)
3: parseTrees = Parsing(tokenised)
4: EARSsentences = Mapping(parseTrees)
```

Freely available and good enough tools exist for tokenising, tagging and parsing. Preprocessing and mapping to EARS, however, had to be designed, built and tailored for the purpose of handling the specific input data.

The EARS Converter is meant for SRS data. The jargon between the SRS of different systems varies a lot, which makes it challenging to optimise the tokenising and tagging specifically for all different systems' SRSs. Because of that, only more general purpose tokeniser, tagger, and parser are applied in this implementation of the EARS Converter.

It is possible to build the EARS Converter with different configurations. In the configuration implemented in this thesis, the tokenising is done by the NLTK's English tokeniser and the tagging and parsing are handled by Stanford parser. Mapping to EARS is implemented making use of the parse trees provided by the parser. The preprocessing phase also applies the tokeniser.

The EARS Recogniser was developed to help assessing the EARS Converter. The EARS Recogniser recognises and labels sentences by what EARS pattern they represent, if any. Figure 13 presents the only functionality the EARS recogniser has: recognising EARS patterns.

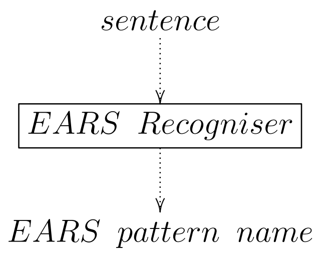


Figure 13. The EARS Recogniser. The Recogniser takes a sentence as an input and outputs the name of the EARS pattern the sentence's structure represents. If the sentence is not in EARS form, the Recogniser outputs UNKNOWN.

The EARS Recogniser takes into account only whether or not the sentence contains the EARS pattern structure superficially. It does not examine the connections between the words and their purpose. It only takes into account the structure words and how many other words are in which positions between, before, and after the structure words.

This chapter is arranged in the following way: The first section discusses the limitations the implementation of the EARS Converter faces. The second section describes the preprocessing phase. In the third section is described how the tokenising is handled. In the fourth section, the tagging and parsing tasks are explained. In the fifth section is presented how mapping to EARS is implemented in the EARS Converter. In the last section is presented the EARS Recogniser, a small algorithm that can be used to check whether a sentence conforms to EARS patterns.

3.1. Limitations

SRS documents as the target data limits the kind of processing the data should get. Many different kinds of SRS documents exist. Some are casually written when others follow strict patterns. Some describe a very complex software, whereas some may depict only a simple system. The SRS documents to be analysed by the EARS Converter had to be further limited to specific kind of SRS documents. English language requirements were chosen as the only expected input, since it is a simple format compared to whole documents and because English is a common language in software development.

Since natural language is so ambiguous, there are many kinds of ambiguities that could be attempted to remove from it. Structural ambiguity was chosen as the main target of ambiguity reduction. It is done by converting NL requirements into EARS patterns using NLP techniques. The applied NLP techniques were chosen based on their popularity, applicability and usefulness.

This section discusses the limitations of the implementation of this thesis. The rest of this section consists of three parts. The first part focuses on the limitations SRS data brings. The second part explains some of the limitations the NLP tools have. In the last part some limitations for ambiguity reduction with the implementation are discussed.

3.1.1. Software Requirements Specifications Data

SRS data in English is not a specific enough scope by itself and it was specified further. Gathering SRS documents proved to be a challenging task due to privacy issues, and thus testing the application had to be done using alternative data, which resembles SRS data. Only one language was chosen as the target language, since other languages would have required as much expertise and material as English and those were not available at the time of writing the thesis.

Single requirements written in English language were chosen as the target input. That way the preprocessing phase of the EARS Converter is better equipped to handle requirements coming from different storage formats and styles.

Availability

One of the largest challenges faced during writing the thesis was the unavailability of public SRS documents. Despite the vast amount of SRS documents in the world, it is hard to come by many of them at the same time. The majority of SRS documents are owned by companies that may wish to keep them private. Unfortunately, gathering a huge database of SRS documents is out of scope of this thesis.

High quality benchmark data for SRS does not seem to exist yet. Some have attempted to gather a database of SRS data, but the attempts have not yet been very successful. For example, nlrpBENCH [70] was launched around 2014 and its database does not seem to have been updated since 2015 (10/2018, http://nlrp.ipd.kit.edu/). Despite the lack of recent updates, some of nlrpBENCH data was used for testing the implementation, since no better options were available.

For the sake of simplicity, the chosen target SRS documents contain the requirements in one document. The requirements are also named. In the preprocessing, other data than the requirements and their names is discarded. That includes background information chapters, pictures and tables.

Limiting the target data to SRS requirements only reduces the freedom of the language so that the structure can be monotonous and each sentence is expected to contain important information. Natural language in free form can contain more sentences with less importance and more variety. For example, technical documents rarely have need for quotations, since they do not describe events the way i.a. novels do.

Language

This thesis only handles requirements written in a specific natural language, English. The EARS methodology is based on English, and to the best of the author's knowledge, it is not known whether or not it could be easily translatable to another language.

The multitude of ways NL can be used forces the algorithms developed for the solution to be flexible about the language. Because of the limited amount of uniform requirements data, statistical learning solution ideas were mostly discarded. Handling NL reliably through machine learning methods demands a relatively large amount of data to teach the system and to test it.

A lot of SRS are written with tables and pictures mixed in with NL. The EARS Converter does not interpret tables nor pictures, which leaves out important information when they are present in the data. When those are present in the data and it is ran through the EARS Converter, it is recommended to use the tables and pictures as is without trying to convert them. They are not in NL, and EARS fits only with NL.

3.1.2. Natural Language Processing Techniques

It might be possible to enhance the implementation with more NLP tools, but to create the very basic version of it, only a few tools were required. The implementation requires tokenising, tagging and parsing in addition to fitting into EARS patterns and preprocessing the data.

Despite its popularity for tasks that resemble the EARS Converter, named entity recognition (NER) was not implemented for tagging or otherwise due to its potential for complicating the processing. Also, it could be difficult to find a ready made NER-tagger that would work well with the jargon used in the SRS documents of different fields of software development.

Stop-word removal was not implemented either. If the system was meant for getting a general idea of the requirements, stop-word removal could be a powerful tool when trying to get the main gist of the requirements. However, since the system relies on sentence structure, and stop-word removal typically removes words that define structure, applying stop-word removal was deemed unnecessary and possibly even harmful for the system.

The parser of choice for the Converter was Stanford parser [23]. The parser has been trained on Penn treebank corpus, which is one of the largest datasets that can be used for such task. [21]

One problem with Penn Treebank corpus is that it does not include technical software documents and so its vocabulary does not cover a lot of jargon used in SRS. Another problem is that the corpus has not been annotated for all kinds of purposes and some of the annotations are ambiguous or incorrect (although most of them are fine). [71] Because of these problems, the parser may also annotate some technical specifications incorrectly. For example, Figure 14 demonstrates how a parser may have incorrectly tagged words and created an incorrect parse tree when the training data does not include enough of the jargon used in the sentence.

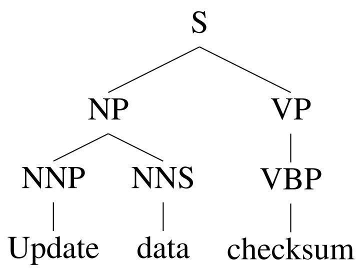


Figure 14. A parse tree of the example sentence: "*Update data checksum.*" The parse tree was produced by Stanford parser, which has been trained on data from the Penn treebank. In the correct annotation "Update" should be the main verb and "data checksum" should be a noun phrase, where "checksum" is the target to be updated.

Due to time constraints of the project, no other parsers were used. The EARS Converter could, at least in theory, use other parsers as well, as long as their tags fit the implementation otherwise.

3.1.3. Ambiguity and Complexity Measurement

The EARS Converter does not measure the complexity and ambiguity of the sentences by itself. For testing purposes such measures are used, however. The EARS Converter attempts to decrease structural ambiguity by forcing a structure on the sentences.

There is a chance that reducing the structural ambiguity by pruning and rearranging the sentence structure trees actually causes other kinds of ambiguity. For example, parts of sentences that contain essential information for their interpretation may be lost. Because of this, the Converter in its current form is recommended for being used as a helper for those who write requirements or want to clarify them, instead of as an independent interpreter.

3.2. Input and output

There are a lot of different SRS documents, written in various styles and formats. For EARS Converter it means that the preprocessing needs to be tailored manually for each type of SRS. Before the data is preprocessed for the EARS Converter, it has to be analysed so that it will reach the correct format in the intended way.

Algorithm 2 describes the preprocessing in general. Different SRS documents can have their requirements described in unique ways. Because of this, the preprocessing ensures that the data is in a suitable format for the Converter.

Algorithm 2 Preprocessing

```
Input: NL data
Output: Preprocessed data
1: limit the data to desired requirements
2: if NL data arranged in requirements then
3:     ensure that each requirement has a name
4: else
5:     arrange data in requirements
6:     name the requirements
7: end if
8: return named requirements
```

The basic processed data format is raw text, or .txt files. Raw text data is easy to handle and it does not contain extra pictures, unless the data happens to contain ascii pictures or other kind of word art the preprocessing did not take into account. The preprocessing should remove all kinds of tables and word art, though.

3.2.1. Input Data

The EARS Converter is meant for SRS documents only. The preprocessing can be done on many different kinds of data, but it is easiest if it is in .txt file format and does not contain any pictures or diagrams, just natural language. In addition to file input, the EARS Converter can also process single sentences.

Format

Other data formats besides raw text files were considered as the optimal input, but due to the simplicity and flexibility of the .txt file type, it was chosen.Some of the input data may be in the form of .doc or .pdf files but the text needs to be extracted from them during or before preprocessing so that the preprocessed file can be in the correct form.

Style

The styles of the possible input documents vary a lot. The allowed styles depend on how much effort can be applied in the preprocessing phase. An optimal input document for the EARS Converter would already contain the requirements so that each line represents a requirement and begins with the requirement name, which does not contain space characters, and continues with the NL requirement description sentences.

3.2.2. Preprocessed data

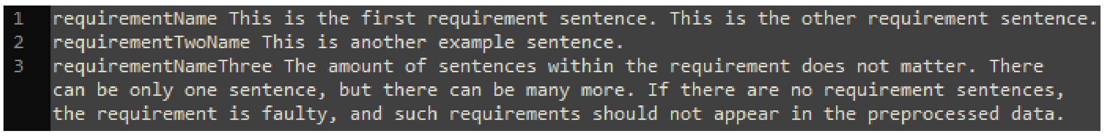
After the data has been analysed, it has to be formatted so that the tokeniser can understand it. In the preprocessed format, the data is given to the tokeniser to process. If the data has been preprocessed incorrectly, the tokenising will not fix it and could make it worse. The manual preprocessing must be done with care and documented properly, even if it is partially automated so that the data goes through an algorithm rather than human hands.

Format

The preprocessed data format is a single .txt file. A .txt file only contains raw text data, which is easy to transform into different formats. The EARS Converter only handles raw text data, which is supposedly in English.

Style

The preprocessed style was chosen so that it represents requirements in a simple manner. Each line represents a requirement. The requirement begins with a name followed by the sentences of within the requirement. The EARS Converter separates the name from the requirement by space character and so the name may not include space characters. In Figure 15 is shown an example of what the preprocessed data can look like.



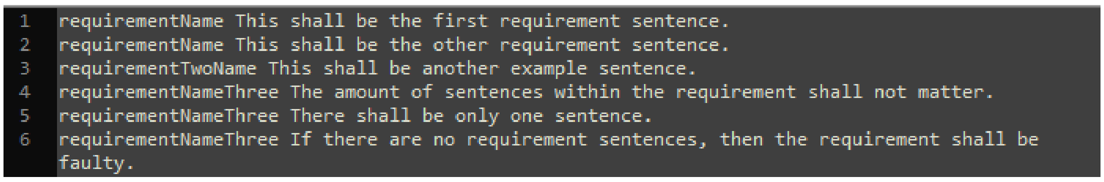
```
requirementName This is the first requirement sentence. This is the other requirement sentence.
requirementTwoName This is another example sentence.
requirementNameThree The amount of sentences within the requirement does not matter. There can be only one sentence, but there can be many more. If there are no requirement sentences, the requirement is faulty, and such requirements should not appear in the preprocessed data.
```

Figure 15. Preprocessed example requirements as seen on a text processor. Each of the three lines represent one requirement, and the first word of each line is a requirement name.

3.2.3. Output data

When the input of the EARS Converter is a file, it forms an output file that contains the transformed sentences. In the case where the EARS Converter input is given manually instead of as a file, it does not form a summary file of all the sentences. It outputs the transformed sentences on command prompt.

For each input data sentence, the EARS Converter forms an equivalent transformed sentence, if possible. In the output file the sentences are arranged so that each sentence is in a separate line, and the requirement name is at the start of the line. If the sentence could not be transformed, its line only contains the name of the requirement the sentence belongs to. In Figure 16 is an example of how the output sentences are arranged in a file.



```
requirementName This shall be the first requirement sentence.
requirementName This shall be the other requirement sentence.
requirementTwoName This shall be another example sentence.
requirementNameThree The amount of sentences within the requirement shall not matter.
requirementNameThree There shall be only one sentence.
requirementNameThree If there are no requirement sentences, then the requirement shall be
faulty.
```

Figure 16. An example of requirements that have gone through the EARS Converter. See preprocessed version of the same sentences in Figure 15.

3.3. Tokenising

Single requirements were extracted from requirements specifications. One requirement can contain several sentences. The sentences were tokenised as separate units and the words within the sentences as separate units within the sentences. This way they are easier to handle through different NLP techniques. Algorithm 3 describes the tokenising process.

Algorithm 3 Tokenising

```
Input: Preprocessed data
Output: Tokenised data
1: interpret each line in the data as requirement
2: for all Requirements do
3:     Separate sentences
4:     for all sentences do
5:         Separate words
6:     end for
7: end for
8: return Requirements, sentences in requirements, words in sentences
```

3.3.1. Requirements

An important part of tokenising happens already in the preprocessing phase. In the preprocessing phase, the requirements are arranged in their own separate lines in a text document and named if they did not already have names. In the tokenising phase, the requirements are simply read line by line.

The input SRS vary in style and format and some may not even have requirements separated properly. If they are not separated properly, there are two options: Either analysing and separating them manually, or treating each sentence or paragraph as a separate requirement. However, it is recommended that in the input data the requirements would be modular, originally separated into individual requirements.

3.3.2. Sentences

In natural language text, sentences are not always annotated in the form where the sentence begins with a capital letter and ends with a period. Not even in English language text. There can be periods that mark abbreviations or are otherwise part of a name or a number. Also, there are different punctuation marks that can end a sentence, and sometimes natural language writers may forget to mark the end of sentence, especially when the next character is a next line character or there is no next character.

To make it more likely that the sentences are separated correctly from each other NLTK sentence tokeniser was used. [29, 30]

3.3.3. Words

To get the POS-tags, the words had to be tokenised first so that each word could be given a POS-tag. In English language, if the clauses have been separated from each other, it is simpler to separate words from each other, since a dot at the end of word marks an end of sentence or an abbreviation, and when the domain is just one sentence, there can be only one end of sentence and others must be abbreviations.

Like with sentences, the EARS Converter uses NLTK for tagging words as well. [29, 30] The NLTK tokenised used is the "TreebankWordTokenizer" which has been made for tokenising to words sentence by sentence.

3.4. Tagging and Parsing

The EARS Converter applies part-of-speech-tagging (POS-tagging) and parsing for creating its main analysis structures, the sentence trees. A sentence tree describes how a sentence's words can be divided into phrase chunks. A sentence can have multiple different possible trees, but a parser usually provides only one possibility based on the algorithms it uses and the POS-tagging of the sentence.

Parsers are used for finding useful chunks of parts of sentences. Unfortunately, an optimal parser for SRS data does not exist yet, and creating such would require a lot of SRS data to train it. Thus, creating a parser of our own is out of scope of this thesis. Instead, Stanford parser was chosen, due to its popularity, availability, and the large amount of data that was used with training it.

Different kinds of taggers exist, but only a POS-tagger was necessary for the implementation of the EARS Converter. The POS-tagging of the EARS Converter is done by Stanford parser. With future development of the EARS Converter, different POS-taggers might be used in addition to the one included in Stanford parser.

Algorithm 4 portrays the parsing process of the EARS Converter. The Stanford parser includes tagging in its parsing process, and thus a separate tagging step was not included.

Algorithm 4 Parsing

Input: Tokenised data
Output: Parse trees of each sentence

```
1: for all Requirements do
2:     add requirement name to list
3:     for all sentences do
4:         Build parse tree
5:         add built tree under requirement name
6:     end for
7: end for
8: return List of parse trees for each requirement
```

3.4.1. POS-Tagging

Part-of-speech (POS) tags are useful for identifying the purpose of each token in a sentence. The EARS Converter applies POS-tagging to sentences to identify the token types and to have a base to build parse trees on. The EARS Converter uses Stanford parser [23] for both POS-tagging and building parse trees.

3.4.2. Phrase Chunks

Sentences consist of phrases and phrase chunks are phrases when they are handled as units. Parsers that attempt to get phrase chunks without otherwise analysing the sentence structure exist [5] but Stanford parser also tags the word tokens and analyses the sentence structure deeper than just phrases. Stanford parser also builds phrase chunks and trees out of phrases.

Sometimes it is more effective to use phrase chunks instead of phrase trees [5], but in this thesis phrase trees were used instead. Phrase trees contain more information than just phrase chunks. Phrase trees arrange the contents of large phrase chunks into smaller chunks, creating multiple levels of chunking.

Verb Phrases

Verb phrases describe action. They consist of a verb and words that are related to it instead of being related to the subject or object. In the EARS Converter, verb phrases are used for finding the action part of the EARS pattern structure. In Figure 17 is shown an example of what verb phrases can be like.

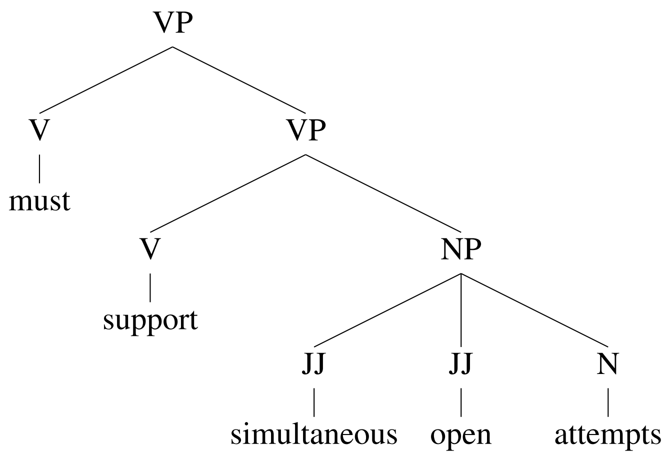


Figure 17. A tree depiction of the verb phrase (VP) in the example sentence from Dataset 2: *"A TCP must support simultaneous open attempts."* "Must" and "support" are both verbs (V) and "simultaneous open attempts" is a noun phrase (NP) part of the verb phrase.

Noun Phrases

Noun phrases describe a thing, an object, or an actor. EARS Converter applies noun phrases mainly in finding the actor. Noun phrases consist of a main noun and words related to it. In Figure 18 can be seen a couple of examples of noun phrases.

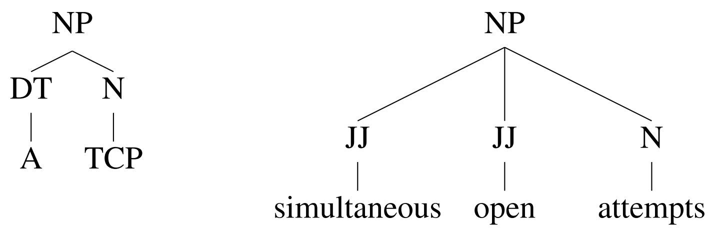


Figure 18. A tree depiction of the noun phrases (NP) in the example sentence from Dataset 2: *"A TCP must support simultaneous open attempts."* "TCP" and "attempts" are both nouns (N), "A" is a determinant (DT) and "simultaneous" and "open" are adjectives (JJ).

3.4.3. Main and Subordinate Clauses

Subordinate clauses describe information closely related to the main clause's topic. With requirements specifications, a subordinate clause could describe a condition under which the main clause happens. Finding subordinate clauses within the sentence can be the easiest way to find EARS pattern conditions.

3.4.4. Stanford Parser

The Stanford parser is a statistical parser. It learns from the data it is fed, which, in the case of the implementation in this thesis, is the Penn Treebank corpus. Because of this, the data dictates a lot of what it can do. The parser produces POS-tagged data and parse trees that can have multiple levels. The POS-tagged text is sorted into phrase chunks and the phrase chunks are further arranged into higher level phrase chunks.

3.5. Mapping to EARS

During the writing of this thesis, the author designed and built a simple EARS Mapper specifically for the thesis project to map the extracted phrases into EARS pattern structures. It relies on the input data to make sense and be correctly POS-tagged and built in a tree form (e.g. Figure 19). The focus of the EARS Mapper is on the structure of the sentence that it extracts from the tree form.

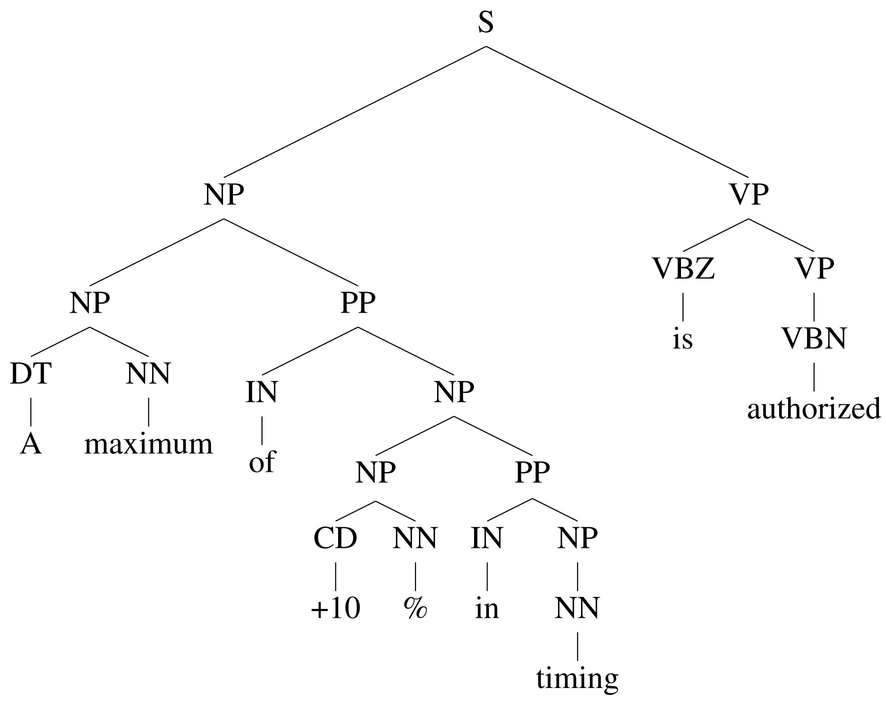


Figure 19. A parse tree of the example sentence: "*A maximum of +10% in timing is authorized.*" The sentence has a clear structure. It consists of a noun phrase (NP) and a verb phrase (VP). It easily fits in the Ubiquitous EARS pattern and becomes: "*The maximum of +10% in timing shall be authorized.*"

The basic structure of the Mapper is described in Algorithm 5. The Mapper requires at least a main VP to be able to build any of the EARS patterns. If a main NP is not found, the Mapper gives the sentence one ("The OS"). To find the conditions of EARS patterns, the Mapper applies a list of marker words that can be used to define whether the subordinate clause is a potential EARS condition.

Algorithm 5 Mapping

```
Input: Parse trees
Output: EARS form requirements specifications
1: for all Parse trees do
2:     Find main VP
3:     if main VP is found then
4:         Find main NP
5:         if main NP not found then
6:             add "The OS" as the main NP
7:         end if
8:         Build base sentence using main VP and main NP
9:         Find subordinate clauses from tree
10:        if One or more subordinate clause found then
11:            for all subordinate clauses do
12:                Find condition marker words
13:                if Potential marker words found then
14:                    build condition
15:                end if
16:            end for
17:            for all condition do
18:                add condition to base sentence
19:            end for
20:        end if
21:    end if
22:    if A sentence was built then
23:        finish EARS sentence
24:    end if
25: end for
26: return Requirements in EARS form
```

The Mapper takes sentence trees as input and outputs EARS sentences. If the sentence is not in EARS form, the mapper recognizes if there are phrase structures that could fit in EARS form present in the sentence. It uses them to rebuild the sentence in EARS form. In Figure 20 is presented the basic idea of the EARS mapper in picture form.

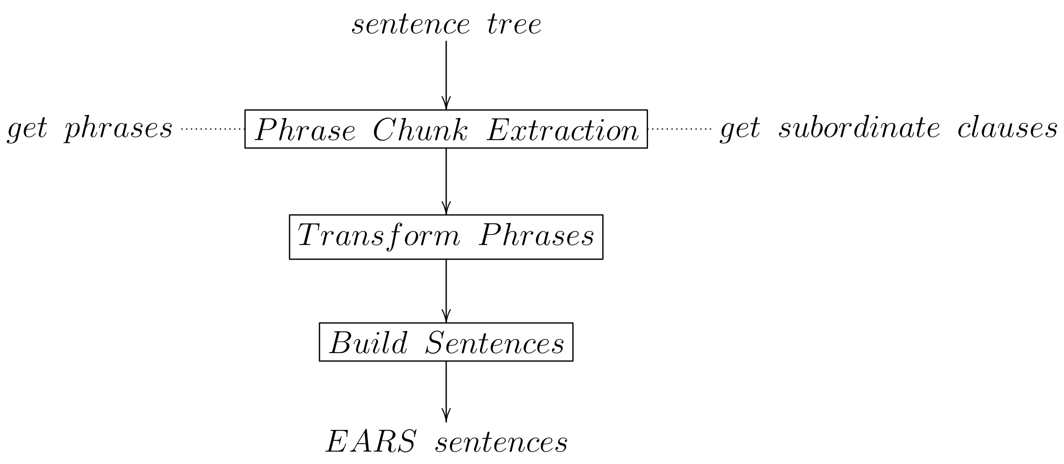


Figure 20. The EARS Mapper. First the relevant phrase chunks and potential extra sentences are extracted from the input tree. Next the phrases are transformed in the form they would appear in EARS patterns. Lastly, the sentences are rebuilt in their EARS form.

3.5.1. Extracting Phrase Chunks and Subordinate Clauses

Phrases and subordinate clauses are extracted from the sentence tree. The most important phrases are the main noun phrase and the main verb phrase, which can be turned into the actor and the action. The subordinate clauses can often be turned into conditions.

Main Verb Phrase

The main verb phrase represents the main action in the sentence. It can vary from a complicated description, like "iterates over a wide dataset called Example Data", to a single verb, like "exists" or "closes". Without a main verb phrase, a sentence is not complete.

The Mapper transforms the original sentence's main verb phrase into its EARS form if it is not in it already. In EARS patterns, the main verb phrase starts with "shall" followed by a verb in its base form (e.g. "exist", "close", "be") and its modifiers, like the object or descriptions of how the verb is executed.

If the sentence structure does not have a verb phrase, it is likely that the sentence has not been constructed properly. In English, a proper sentence requires a verb and for it to be possible to build an EARS pattern, the sentence requires a verb phrase to have an action that shall be implemented. If the main verb phrase is not found, the EARS converter warns that an EARS pattern could not be produced.

Main Noun Phrase

In SRS sentences, the main noun phrase should define which part of the system does the action. The EARS mapper recognizes the main noun phrase as the noun phrase that is in the same level as the main verb phrase.

If there is no main noun phrase, but a main verb phrase exists, a main noun phrase ("the OS") is added by the mapper. In Figure 21 is shown the structure of the added noun phrase when one is not found.

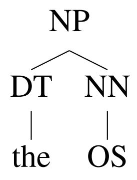


Figure 21. The "the OS" NP, which is added if a main NP is not found.

In some cases the found main noun phrase is not the actual subject of the sentence. For example, if the sentence is in passive form (e.g. "there shall be sunny") the main noun phrase (e.g. "there") is not the actual subject. That kind of sentences can be easily translated into EARS form, but the meaning of the sentence may not become more specific.

Subordinate clauses

The conditions within EARS form sentences are subordinate clauses. The Converter extracts subordinate clauses from the original sentences to check if they already are in EARS condition form or if they could be transformed to be in EARS form.

During the development of the EARS Converter, it was noticed that the parse trees often had subordinate clauses that had subordinate clauses, even though they could be represented as two separate subordinate clauses. Thus, it was decided that if a subordinate clause is within another subordinate clause, the EARS converter attempts to separate them from each other.

Some subordinate clauses are already in the form of EARS conditions. The only modifications the EARS converter does to those is separating subordinate clauses from them. If they do not have a comma where it should be according to the EARS pattern, it adds one.

3.5.2. Transforming Phrases

In the phrase transforming phase the processed phrase chunks are evaluated by how well they would fit EARS patterns. The optimal EARS pattern structures are NP-VP and SBAR-NP-VP, where SBAR can be multiple SBARs. The transformation idea is depicted in Figure 22,

$$\begin{tikzcd}
SBAR \arrow[d, dotted] & NP \arrow[d, dotted] & VP \arrow[d, dotted] \\
condition & actor & action
\end{tikzcd}$$

Figure 22. The EARS patterns follow the formula where a condition for the action to happen is placed before the actor and the action after the actor. Conditions are marked by the the words "while","when","where" and "if..., then" and in Ubiquitous pattern the condition does not exist. The actor is labeled with "the" and the action with "shall."

To fit the EARS patterns, phrases are extracted from the sentences. The basic structure of an EARS pattern is a condition followed by the actor and the action that the actor shall do given the condition. Ubiquitous pattern contains only the actor and action. Within the patterns, the actor is marked with "the", the action with "shall" and the condition by "when", "while", "where" or "if..., then."

Action

Actions are main verb phrases that have been transformed to fit into EARS patterns. In some cases, the verb phrase (VP) can already be in the correct form (begins with "shall") and it does not need any transforming. The EARS Mapper does not check linguistic correctness of the sentence, however, so the original one can be faulty, which leads to the transformed one to be faulty as well.

If the main verb is in simple past tense, the EARS Converter attempts to transform the phrase to fit into EARS pattern, but it does not know how to transform the verb tense. This, however, should not cause too much trouble, since simple past tense is rare as the main verb of SRS documents. Requirements describe what the system is supposed to do, instead of what it did.

When the VP begins with an auxiliary verb (can, could, will, would, shall, should, may, might, must, do or does), the EARS Converter ensures that "should" is always in its place. The easiest VPs to transform into the EARS action form are those with auxiliary verbs, since they, if written correctly, already are in the correct form otherwise.

Different forms of "is" (is, ’s, are, was, were, am, ’m) in the input main VP sentence are transformed into "shall be" by the EARS converter.

Actor

Actors are noun phrases that fit in the EARS pattern’s ubiquitous part’s first slot. Ideally, the actor is the subject of the sentence, but some types of sentences do not have an actor within the sentence structure (e.g. imperative sentences, where the actor is the one or ones who the sentence is directed at)

The phrases found in natural language SRS documents are relatively often in imperative form (e.g. "run away", "update slot number") when the actor is not present within the sentence. It was decided that in the case where the actor is not found, the mapper adds an actor, "the OS", as shown in Figure 23.

			Run	*the*	*script*
The	*OS*	*shall*	*run*	*the*	*script*

Figure 23. An example of an imperative form sentence to Ubiquitous EARS pattern sentence. The imperative form sentence does not contain the actor within itself. To make it an EARS form sentence, the actor ("The OS") has to be added in addition to transforming the main verb to create the action part. In this case, the transforming consists of adding "shall" at the beginning of the action and leaving the main verb ("run") in its base form it appeared in.

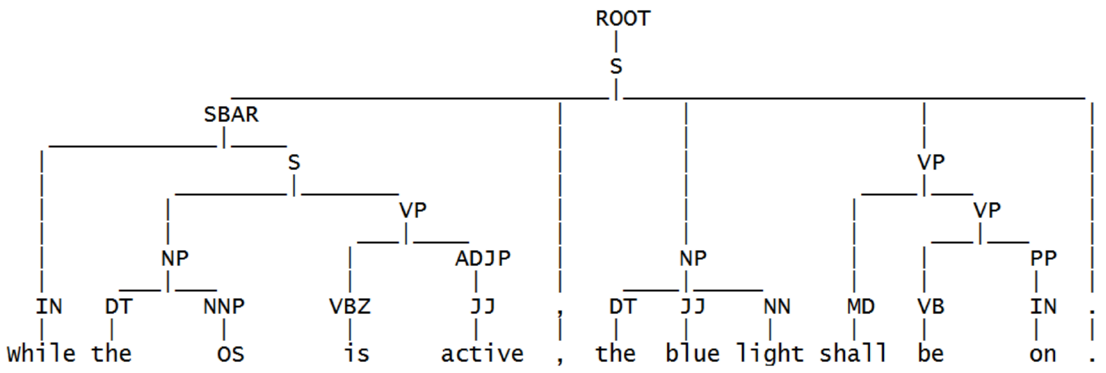
Condition

The condition is a subordinate clause that fits into an EARS pattern that is not ubiquitous pattern. The condition begins with one of these words: when, while, where, if. If the condition begins with "if" it ends with "then." Otherwise the condition's last word has not been strictly determined. A combination of patterns may contain several conditions with one Ubiquitous pattern followed by them.

3.5.3. Fitting to EARS

The main structure of most of the EARS patterns consists of a subordinate clause and a main clause. The main structure of the Ubiquitous pattern is the only one that consists of only a main clause, and complex EARS patterns can have more than one subordinate clauses in their main structure. The input sentences may not follow the EARS pattern structure, so the sentences are divided into specific parts that can be used for trying to fit the sentence, or at least parts of it, into EARS pattern structure.

Phrase chunks and subordinate clauses can be used for building EARS patterns. In Figure 24 is shown an example sentence tree, where main verb phrase and noun phrase are clearly presented along with a subordinate clause.



```
                                          ROOT
                                           |
                                           S
                ___________________________|_________________________________
              SBAR                     |        |                |           |
   ____________|____                   |        |                |           |
  |                 S                  |        |                VP          |
  |         ________|_______           |        |            ____|___        |
  |        |                VP         |        |           |        VP      |
  |        |             ___|____      |        |           |     ___|___    |
  |        NP           |       ADJP   |        NP          |    |       PP  |
  |     ___|___         |        |     |    ____|_____      |    |       |   |
  IN   DT     NNP      VBZ       JJ    ,   DT   JJ    NN    MD   VB      IN  .
  |    |       |        |        |     |   |    |     |     |    |       |   |
while the      OS       is     active  ,  the  blue light shall  be      on  .
```

Figure 24. State-Driven pattern sentence tree as produced by the EARS Converter in ASCII form. The main level, the first branching point, contains an SBAR, a punctuation mark, an NP, a VP and another punctuation mark. The subordinate clause (SBAR) conforms to the SDP condition style, the NP to actor style and the VP to action style.

The Ubiquitous Pattern

The Ubiquitous pattern (UP) is the simplest of the EARS patterns and it is included in all the other patterns. Thus, it took the first priority while implementing the Converter.

The Converter interprets the sentences in parse tree form produced by Stanford parser. In Figure 25 is shown how NLTK with Stanford parser depicts a sentence in the Ubiquitous pattern form. The Ubiquitous pattern contains two parts, a noun phrase (NP), and a verb phrase (VP).

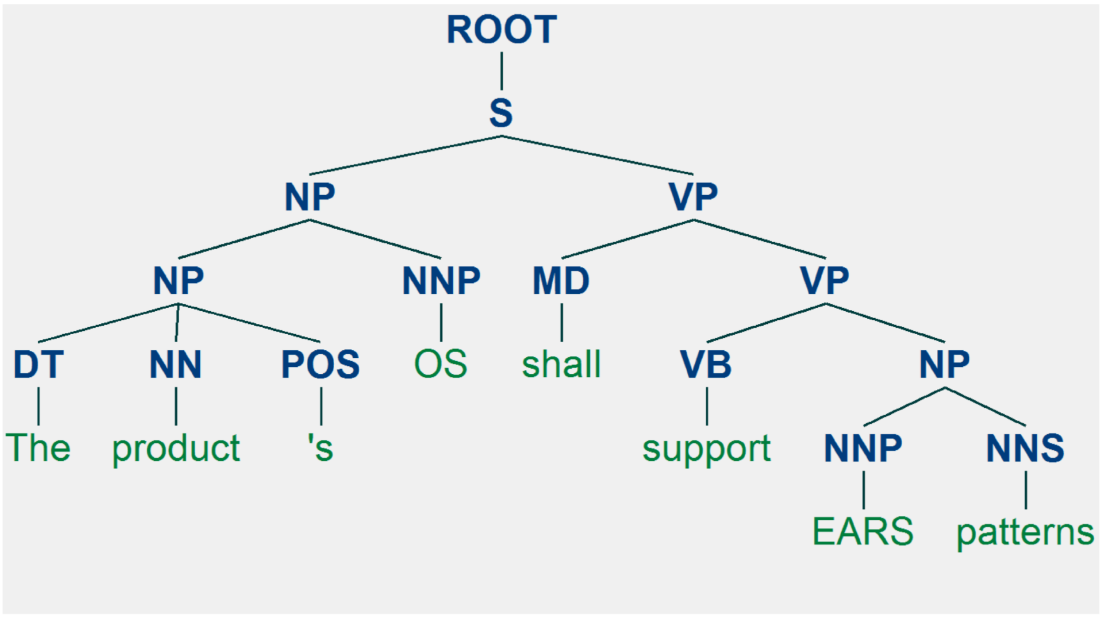


Figure 25. A sentence tree as produced by Stanford parser with NLTK. The sentence, "The product's OS shall support EARS patterns," is in Ubiquitous EARS pattern form, since it begins with a noun phrase which begins with "the" and continues with a verb phrase beginning with "shall".

In Figure 26 is presented the mould in which the NP and VP should set for Ubiquitous pattern to be produced. For creating an UP sentence that contains useful information, the NP has to be the actor and the VP needs to be the action the actor carries out. The EARS mapper does not take a stand on whether the actor and action make sense. If the NP and VP in the sentence do not make sense, the new Ubiquitous pattern may not make sense either.

$$\boxed{The - NP \qquad shall - VP,}$$

Figure 26. The Ubiquitous pattern build consists of an actor and an action. The actor is a noun phrase, that has been transformed so that it begins with "the" if it did not originally. The action is a verb phrase, that has to begin with "shall".

The Event-Driven Pattern

The Event-Driven pattern (EDP) consists of a condition, an actor and an action. The condition begins with the word "when" and the actor and action are similar to the ones in UP. The basic construction of an EDP is shown in Figure 27. The EDP describes an action done by the actor following the event described in the condition.

$$\boxed{When - SBAR, \qquad the - NP \qquad shall - VP,}$$

Figure 27. The Event-Driven pattern build consists of a condition, an actor, and an action. The condition is a subordinate clause beginning with "When" and the actor and action are similar to the ones in the UP.

The Event-Driven pattern (EDP) may be chosen to contain Unwanted Behaviour pattern (UBP) as well. By default, the EARS Mapper does so, except in the situations where the potential condition begins with "if", which is a UBP condition's marker word.

The State-Driven Pattern

Like EDP, State-Driven pattern (SDP) also consists of a condition, an actor and an action. SDP describes an action that the actor does in the state described in the SDP condition. The marker word for SDP condition is "while". Figure 28 shows the basic structure of the SDP.

$$\boxed{While - SBAR, \qquad the - NP \qquad shall - VP,}$$

Figure 28. The SDP build's condition is a subordinate clause (SBAR), that begins with "While". The actor and action of SDP follow the same style as the other patterns.

The Optional Feature Pattern

The OFP condition begins with "where". Figure 29 shows the structure in which the OFP is built.

$$\boxed{Where - SBAR, \qquad the - NP \qquad shall - VP,}$$

Figure 29. The OFP build's condition begins with "Where" and it otherwise is similar to the other pattern builds.

The Unwanted Behaviour Pattern

The Unwanted Behaviour Pattern (UBP) is the most complicated one of the single patterns, because it contains four marker words instead of three or two. The EARS Converter typically interprets potential conditions that could be UBP conditions as EDP conditions, unless they already contain at least half of the UBP condition structure (either "if" or "then").

As with other patterns, besides the UP, the UBP consists of a condition and UP pattern after it. The marker words for the condition are "if" and "then." In a grammatically correct UBP sentence, there is also a comma before "then", which is added to the condition when it is built. In Figure 30 is shown the structure of how the UBP is built.

$$\boxed{If - SBAR, -then \qquad the - NP \qquad shall - VP,}$$

Figure 30. The UBP pattern build's condition is different from the other pattern builds in that it has to end with ", then" in addition to beginning with "If". Otherwise it is very similar to the other pattern builds.

Combinations of Patterns

If the EARS Converter detects more than one subordinate clause, it attempts to create more than one condition. If they are nested, it first separates them and builds conditions out of all of them, if possible. However, if the pattern does not contain a main VP, building a combination pattern is not possible, as it is not possible with any other pattern.

Unknown Pattern

When the EARS Converter does not find the necessary main VP, the pattern is unknown. In this case, the Converter outputs a single line "-" instead of a sentence, in addition to giving a warning.

3.6. Recognising EARS Patterns

The EARS Recogniser is a separate unit that can be used to check whether a sentence conforms to any EARS pattern structure. It takes a sentence as an input and checks if it has the marker words in place with the correct number of words in between them.

The Recogniser was created for having a tool that is independent of the Converter and can be used for testing. In contrast to the Converter, the Recogniser only goes into word level. It does not matter what words are in between the EARS structure elements, as long as there is the right number of them and they do not break the structure.

3.6.1. Structure

The Recogniser mainly consists of only two parts: the tokeniser and the template checker. The tokeniser is the same one used with the EARS Converter. The template checker compares the tokenised sentence's structure against EARS pattern structures. Figure 31 shows the main structure of the Recogniser.

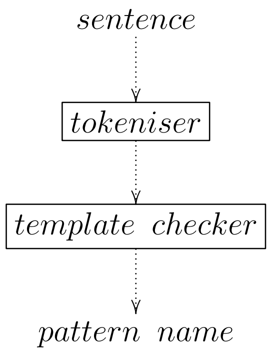


Figure 31. The EARS Recogniser.

Algorithm 6 describes how the EARS Recogniser works. The tokeniser is required for separating different words from each other so that the sentence structure can be better analysed. The marker words are the words that appear in EARS form sentences (e.g. shall, while) and they form the pattern structure. The numbers of non-marker words are calculated, because in EARS patterns, there has to be a certain number (0 or >0) of non-marker words between each marker word.

Algorithm 6 The EARS Recogniser

```
Input: sentence
Output: EARS pattern name or UNKNOWN
1: tokenise sentence to words
2: label marker words
3: count non-marker words between marker words
4: compare against EARS patterns
5: return pattern name
```

3.6.2. Patterns

Each EARS pattern has its own structure. The template checker of the EARS Recogniser extracts the structure of a sentence from the tokenised sentence and compares the structure against the different possible EARS pattern structures.

The structure of the tokenised sentence is extracted as follows: First the non-word tokens are omitted. Then the word tokens' letters are capitalised. Next the words between, before, and after, marker words, are calculated and the structures are built out of the marker words and the number of words between them. Since "the" is not commonly used as a marker word, extra "the"s are removed from the structures leaving only the "the"s that accompany "shall"s. The process is presented in Table 3 step-by-step by transforming an example sentence into a potential pattern structure.

Table 3. Transforming a sentence into a potential patterns structure

Input sentence:	When the process starts, the sentence shall be transformed into a structure.
1. Punctuation removal:	When the process starts the sentence shall be transformed into a structure
2. Capitalisation:	WHEN THE PROCESS STARTS THE SENTENCE SHALL BE TRANSFORMED INTO A STRUCTURE
3. Calculating words:	0 WHEN 0 THE 2 THE 1 SHALL 5
4. Removing extra THEs:	0 WHEN 3 THE 1 SHALL 5

Single Patterns

The Recogniser can recognise all the different EARS patterns based on the structure words. The structure words are: when, while, where, if, then, the and shall. Table 4 presents each basic EARS pattern structure. The x in the table represents the number of words between the previous and next slot and it should be larger than zero. WHEN, WHILE, WHERE, IF, THEN, THE and SHALL are the marker words that appear in EARS pattern structures.

Table 4. Summary of the EARS pattern structures

	Condition			Actor		Action	
Ubiquitous	■	■	■	THE	x	SHALL	x
Event-Driven	WHEN	x		THE	x	SHALL	x
State-Driven	WHILE	x		THE	x	SHALL	x
Optional Feature	WHERE	x		THE	x	SHALL	x
Unwanted Behaviour	IF	x	THEN	THE	x	SHALL	x

Combination of Patterns

In this thesis the combination of patterns was defined as a pattern with multiple conditions, an actor and an action. The EARS Recogniser recognises a pattern as a COMBINATION, when it finds multiple conditions before the actor and action. Each condition, however, has to begin with a different allowed word.

Unknown Patterns

When the EARS Recogniser does not recognise a pattern structure, it outputs UNKNOWN. The reasons for not recognising a pattern include insufficient number of EARS structure elements, too many of them without other words in between, or incorrect order of them. Also, if there are not enough other than structure words in the correct places, the pattern is unknown.

4. TESTING

The testing of the EARS Converter was performed on four different datasets. The datasets were chosen based on availability and resemblance to software requirements. The test were chosen so that they would measure the Converter performance and the ambiguity of the input and output data to see whether it had been reduced.

Before the tests were done, the datasets had to be preprocessed. Preprocessing was tailored for each dataset separately, since the datasets, like real SRS documents, are not uniform in their style and format. In the first section of this chapter is described what had to be taken into account while preprocessing each particular dataset.

The performance of the EARS Converter[1] was measured with the help of the EARS Recogniser[2]. The Recogniser recognises which pattern a sentence represents and also if it does not represent any EARS pattern.

The ambiguity reduction was measured with the help of a few different measures. Each of them aim to provide a different view on the structural ambiguity of the sentences. Whether or not a sentence is in EARS is not taken into account when measuring ambiguity, but the datasets were compared against each other before and after they had been converted in their EARS equivalents by the Converter.

This chapter has been divided in four main sections. The first describes the datasets and the preprocessing done to them. The second explains how the performance of the EARS Converter was tested and the results of those tests. In the third section the structural ambiguity of the datasets is analysed from a few different points of view. The final section of this chapter summarises the test results.

4.1. Datasets and Preprocessing

Since finding freely available SRS documents proved to be difficult, four different datasets, that only resemble SRS documents, were chosen. The datasets present different writing styles of requirements. NL requirements can be written in different styles depending on the software they are meant for and who writes them.

Each dataset contains data that defines requirements for different system. Dataset 1 defines the rules of a game. Dataset 2 defines requirements for Internet hosts. Dataset 3 consists of a few parts of requirements for steam-boiler control system. For comparison, a completely different text, a novel, was chosen as Dataset 4.

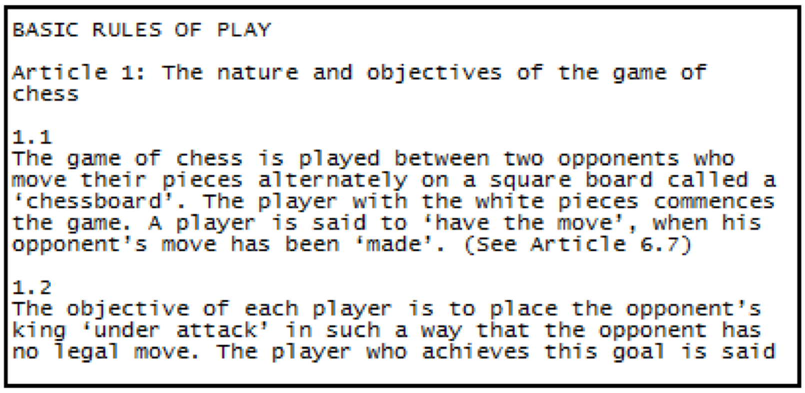
The preprocessing of each dataset follows Algorithm 2 (Chapter 3 Section 3.2). The different structures of the datasets necessitates personalising the preprocessing for each dataset individually. The result of preprocessing is the preprocessed data, which is in .txt format and each line represents a requirement. The first word (non-whitespace characters before any whitespace characters on the line) is interpreted as the name of the requirement and the rest of the line is the contents of the requirement.

[1]Chapter 3 Sections 3.3-3.5.

[2]Section 3.6.

4.1.1. Dataset 1: FIDE Laws of chess

Chess rules were chosen as the Dataset 1 to have a dataset meant to be read by humans with relatively little technical knowledge. FIDE Laws of chess (2014) was listed in the nlrpBENCH website and it was freely available and easy to acquire. [70, 72] The data contains the rules of chess for basic play and competitive play. The extracted data contains only raw text version of the rules in UTF-8 character encoding format. Pictures and special symbols were omitted. In Figure 32 is shown a short excerpt of the data.



```
BASIC RULES OF PLAY

Article 1: The nature and objectives of the game of
chess

1.1
The game of chess is played between two opponents who
move their pieces alternately on a square board called a
‘chessboard’. The player with the white pieces commences
the game. A player is said to ‘have the move’, when his
opponent's move has been ‘made’. (See Article 6.7)

1.2
The objective of each player is to place the opponent's
king ‘under attack’ in such a way that the opponent has
no legal move. The player who achieves this goal is said
```

Figure 32. An excerpt from the FIDE Laws of Chess (2014) from the beginning of the chapter Basic Rules of Play.

FIDE Laws of chess (2014) contains five parts: *Introduction*, *Preface*, *Basic Rules of Play*, *Competition Rules* and *Appendices*. It was decided that the *Basic Rules of Play* and *Competition Rules* were the only parts of the document that would be included in the preprocessed dataset.

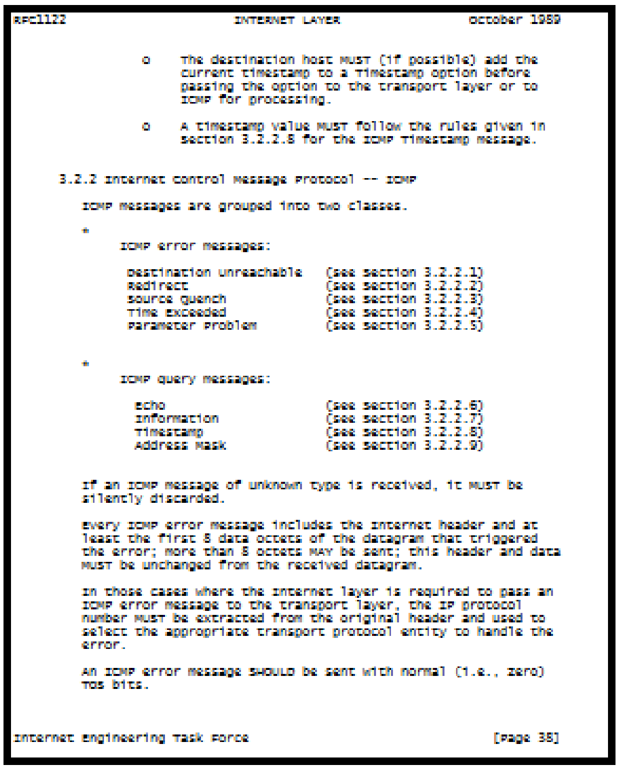
The *Basic Rules of Play* and *Competition Rules* contain numbered parts, "articles", that also contain numbered parts that are titled just by the number. These parts describe the game rules and the competition rules and they resemble software requirements in the sense that they describe what should be done in the game and what should not be done.

The *Introduction* and *Preface* contain only a small number of text and the text describes the nature of the document. The *Appendices* contains text that is arranged in various ways, not just sentences and titles. These parts do not resemble requirements and thus they were omitted when building Dataset 1.

The chess rules' structure is simple and clear and it does not require very complicated preprocessing to make it fit the desired format. First, only the parts *Basic Rules of Play* and *Competition Rules* were extracted. The other parts were discarded. Then, the preprocessing algorithm chose the titles of Article sections' numbered parts as the names of requirements. Finally the sentences following the titles were chosen as the sentences within the requirements and the requirement lines were formed.

4.1.2. Dataset 2: Requirements for Internet Hosts – Communication Layers

Requirements for Internet Hosts – Communication Layers [73] was chosen as the material for Dataset 2. It was chosen because it contains software terminology and SRS can have similar structure. The structure is quite varied, as can be seen in Figure 33. It contains tables and figures in ASCII-form, which had to be omitted when creating Dataset 2. Only text form could be used, but not all of it fit into the desired requirement format easily.



```
RFC1122                     INTERNET LAYER                  October 1989


               o    The destination host MUST (if possible) add the
                    current timestamp to a Timestamp option before
                    passing the option to the transport layer or to
                    ICMP for processing.

               o    A timestamp value MUST follow the rules given in
                    Section 3.2.2.8 for the ICMP Timestamp message.

      3.2.2 INTERNET CONTROL MESSAGE PROTOCOL -- ICMP

         ICMP messages are grouped into two classes.

         *
              ICMP error messages:

               Destination Unreachable   (see Section 3.2.2.1)
               Redirect                  (see Section 3.2.2.2)
               Source Quench             (see Section 3.2.2.3)
               Time Exceeded             (see Section 3.2.2.4)
               Parameter Problem         (see Section 3.2.2.5)


         *
              ICMP query messages:

                Echo                     (see Section 3.2.2.6)
                Information              (see Section 3.2.2.7)
                Timestamp                (see Section 3.2.2.8)
                Address Mask             (see Section 3.2.2.9)

         If an ICMP message of unknown type is received, it MUST be
         silently discarded.

         Every ICMP error message includes the Internet header and at
         least the first 8 data octets of the datagram that triggered
         the error; more than 8 octets MAY be sent; this header and data
         MUST be unchanged from the received datagram.

         In those cases where the Internet layer is required to pass an
         ICMP error message to the transport layer, the IP protocol
         number MUST be extracted from the original header and used to
         select the appropriate transport protocol entity to handle the
         error.

         An ICMP error message SHOULD be sent with normal (i.e., zero)
         TOS bits.


Internet Engineering Task Force                                [Page 38]
```

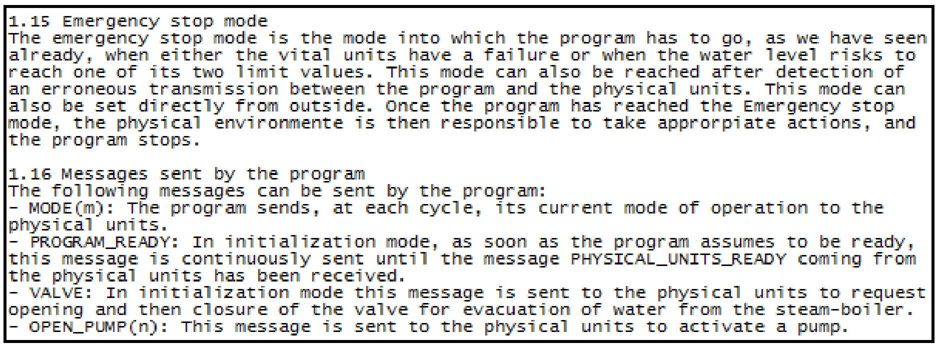
Figure 33. A page of Requirements for Internet Hosts – Communication layers (1989). Note that the file is in .txt format, which does not have actual paging. The pages vary in structure, but this page is a typical example of a page structure.

The requirements were gathered from page 20 to page 107 of the material [73]. Each sentence that contained "may", "should" or "must" was chosen as a requirement sentence, since the document advised that those are its key words for requirements (although only the upper case versions of the words are official, all casings were accepted for Dataset 2). A sentence belongs to a chapter or section and the chapter or section number is its requirement's name.

Dataset 2 is the one of the four datasets that most resembles SRS. The sentences contain software related terms and structures and they have been written in a slightly restricted technical format. Each sentence of Dataset 2 contains a "must", "should", or "may" which can be turned into "shall" to create an EARS structure. Turning them into "shall" form, however, loses some of the sentences' meaning and nuances, because each of them have a specific purpose in the Requirements for Internet Hosts – Communication Layers [73]. However, those meanings are closely related to the meaning of "shall" in EARS sentences.

4.1.3. Dataset 3: Steam-Boiler Control Specification Problem

Dataset 3 contains data in technical description form. The sections 1.10-1.18 of Steam-Boiler Control Specification Problem [74] are the only included data in Dataset 3, making this the smallest of the datasets. Each named message and behaviour in sections 1.16-1.18 are considered separate requirements within the dataset. Figure 34 shows an excerpt from the chosen parts of the Steam-Boiler Control Specification Problem.



```
1.15 Emergency stop mode
The emergency stop mode is the mode into which the program has to go, as we have seen
already, when either the vital units have a failure or when the water level risks to
reach one of its two limit values. This mode can also be reached after detection of
an erroneous transmission between the program and the physical units. This mode can
also be set directly from outside. Once the program has reached the Emergency stop
mode, the physical environmente is then responsible to take approrpiate actions, and
the program stops.

1.16 Messages sent by the program
The following messages can be sent by the program:
- MODE(m): The program sends, at each cycle, its current mode of operation to the
physical units.
- PROGRAM_READY: In initialization mode, as soon as the program assumes to be ready,
this message is continuously sent until the message PHYSICAL_UNITS_READY coming from
the physical units has been received.
- VALVE: In initialization mode this message is sent to the physical units to request
opening and then closure of the valve for evacuation of water from the steam-boiler.
- OPEN_PUMP(n): This message is sent to the physical units to activate a pump.
```

Figure 34. An excerpt from the chosen sections of the Steam-Boiler Control Specification Problem.

4.1.4. Dataset 4: Fiction

As Dataset 4, a piece of literature was chosen as non-technical material for comparison. The chosen text was Pride and Prejudice by Jane Austen. [75] Dataset 4 is the largest of the datasets and also contains the most non-technical language.

The novel is structured like novels often are. It consists of chapters and paragraphs. Unlike with typical SRS documents, the novel contains literary structures, like quotations, when the characters speak and do things. (Some SRS documents might also contain quotations if, for example, they describe use cases where the users speak or write specific things.)

The text file for the novel was acquired from project Gutenberg's achieves [75] and was in .txt form. Thus the preprocessing was rather easy. Each paragraph was considered a requirement and only the chapters of the novel were included without the chapter names.

In preprocessing of Pride and Prejudice, semicolons were replaced by periods to make processing the sentences easier. The Converter makes sentence trees of all sentences. When processing too long sentences to create sentence trees, the NLTK with Stanford Parser gives a warning and does not finish processing such sentences. An example of a sentence, which uses semicolons as links to create an overly long sentence, is the following found in the Pride and Prejudice by Jane Austen (1813) [75]:

> " Every lingering struggle in his favour grew fainter and fainter; and in farther justification of Mr. Darcy, she could not but allow Mr. Bingley, when questioned by Jane, had long ago asserted his blamelessness in the affair; that proud and repulsive as were his manners, she had never, in

> the whole course of their acquaintance– an acquaintance which had latterly brought them much together, and given her a sort of intimacy with his ways– seen anything that betrayed him to be unprincipled or unjust– anything that spoke him of irreligious or immoral habits; that among his own connections he was esteemed and valued– that even Wickham had allowed him merit as a brother, and that she had often heard him speak so affectionately of his sister as to prove him capable of some amiable feeling; that had his actions been what Mr. Wickham represented them, so gross a violation of everything right could hardly have been concealed from the world; and that friendship between a person capable of it, and such an amiable man as Bingley, was incomprehensible."

4.2. Performance

Precision and recall are measures commonly used for measuring the performance of information retrieval systems. The EARS Converter can be thought of as an information retrieval system. The information retrieval measures, precision, recall and F-measure, are calculated out of the recognized and non-recognized EARS sentences. They tell how well the EARS Converter performs by describing how well it turns natural language sentences into EARS sentences.

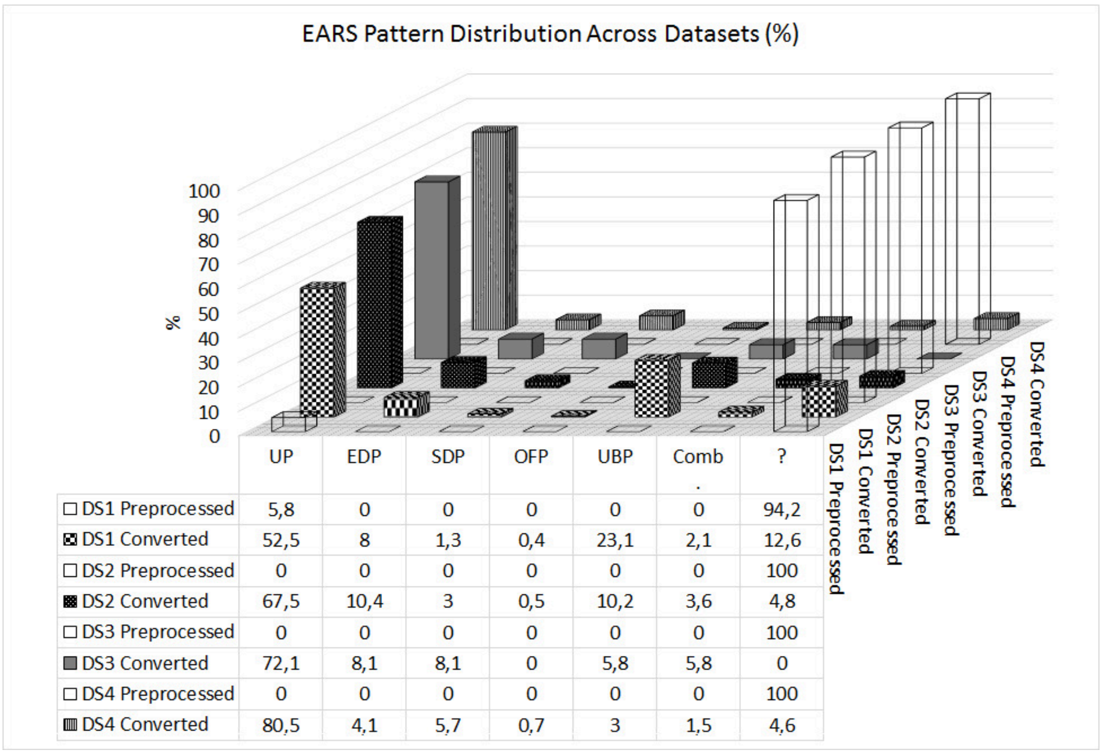
Classifying the sentences is done with the EARS Recogniser algorithm (Chapter 3 Section 3.6). No manual labelling of expected patterns was done, since that would have required more than one available linguistically talented requirements engineer to label the data. Because of the ambiguity of NL, same sentence can be transformed into several different EARS pattern sentences. If only one person determines which EARS pattern a sentence should become, the results are highly biased.

Using the Recogniser the variation in the recognition quality caused by human recogniser is eliminated. However, the Recogniser can not take into account how a human would actually interpret the sentences. Instead, it systematically classifies the sentences based on their surface traits, e.g. which EARS structure words are present. It does not take into account whether the sentence makes sense, only whether it conforms to the EARS patterns.

4.2.1. Pattern Distribution

To evaluate whether the Converter manages to transform sentences into different EARS patterns, the number of each pattern type was calculated over the datasets before and after conversion. The EARS Recogniser was applied to determine which pattern each sentence represents. No manual labelling nor predictions on which pattern a sentence should be after conversion was performed due to time and resource constraints. However, the sentences were manually looked over to ensure they looked valid enough.

In Figure 35 is gathered the percentages of each pattern in each dataset before and after the Converter. Only Dataset 1 contained any recognisable patterns before the converter (5,8% of the sentences) and they were of the basic pattern type: Ubiquitous.



	UP	EDP	SDP	OFP	UBP	Comb.	?
□ DS1 Preprocessed	5,8	0	0	0	0	0	94,2
▩ DS1 Converted	52,5	8	1,3	0,4	23,1	2,1	12,6
□ DS2 Preprocessed	0	0	0	0	0	0	100
▦ DS2 Converted	67,5	10,4	3	0,5	10,2	3,6	4,8
□ DS3 Preprocessed	0	0	0	0	0	0	100
■ DS3 Converted	72,1	8,1	8,1	0	5,8	5,8	0
□ DS4 Preprocessed	0	0	0	0	0	0	100
▥ DS4 Converted	80,5	4,1	5,7	0,7	3	1,5	4,6

Figure 35. The percentages of each pattern type in each dataset before and after the Converter. The transparent bars describe the EARS distributions before the conversion and the non-transparent after the conversion.

After the Converter the percentage of unrecognised patterns by the Recogniser was below 13% in all datasets. The most prominent recognised pattern was the Ubiquitous pattern (UP). Over 50% of all the recognised patterns were UPs in every converted dataset.

Table 5 shows the result sentences distribution in different EARS patterns and unknown patterns (denoted with "?"). Labelling the sentences was done with the EARS Recogniser. The expected order of the different patterns, from most common to least common, is in the rightmost column. It is based on how common the marker words of the different patterns are in English language. Word frequencies were compared against each other in the Corpus of Contemporary American English [76] and British National Corpus [77].

Table 5. EARS patterns in each dataset after EARS Converter

Dataset 1	Dataset 2	Dataset 3	Dataset 4	Expected
UP (52,5%)	UP (67,5%)	UP (72,1%)	UP (80,5%)	?
UBP (23,1%)	EDP (10,4%)	EDP (8,1%)	SDP (5,7%)	UP
? (12,6%)	UBP (10,2%)	SDP (8,1%)	? (4,6%)	EDP
EDP (8%)	? (4,8%)	UBP (5,8%)	EDP (4,1%)	OFP
Comb (2,1%)	Comb (3,6%)	Comb (5,8%)	UBP (3%)	SDP
SDP (1,3%)	SDP (3%)	OFP (0%)	Comb (1,5%)	Comb
OFP (0,4%)	OFP (0,5%)	? (0%)	OFP (0,7%)	UBP

Based on Table 5, the Converter performed rather well on most datasets. The converted Dataset 3 does not have any unknown sentences, but the others have some of them.

Optional Feature pattern is very rare in all datasets, which was unexpected, since "where" is a rather common word. However, "where" is a word that is very commonly used in question sentences, which may not translate into EARS patterns very well.

4.2.2. Recall, Precision and F-Measure

Recall, Precision and F-Measure are all used to assess how well the system retrieved its results. In this thesis, these measures are calculated over all patterns, since it was not possible to forecast which specific sentence would represent which EARS pattern after transformation, only that each sentence should have some EARS pattern sentence equivalent.

Recall describes how much of retrieved information is correct. In equation 7, which describes how recall is calculated in this thesis, *R* is recall, *e* the total number of retrieved sentences (both recognised and unrecognised ones) and *u* the number of sentences that did not get a translation.

$$R = \frac{e}{e+u} \tag{7}$$

Precision describes how much of the retrieved information was also the correct sort of information. With the help of EARS Recogniser, it tells which percentage of the retrieved sentences are in EARS. Equation 8 below describes how precision is calculated. *P* is precision, *r* is the number of recognised EARS sentences and *s* is the number of unrecognised retrieved sentences.

$$P = \frac{r}{r+s} \tag{8}$$

F-measure combines precision and recall. The Equation 5 (Chapter 2 Section 2.1.3) shows how F-measure is calculated.

In Table 6 are listed the results of precision, recall and F-measure calculations for each dataset. Most of the sentences that the converter received as an input were returned in EARS and thus the scores are quite high. For Dataset 3 the Converter returned all of the sentences in EARS form, which resulted in a perfect score for all the three measures. The EARS Recogniser was used to define whether a sentence is in EARS or not.

Table 6. Precision, recall and F-measure of each dataset

	Precision	Recall	F-measure
Dataset 1	0.87	0.99	0.93
Dataset 2	0.95	0.99	0.97
Dataset 3	1	1	1
Dataset 4	0.95	0.92	0.94

The results of Dataset 3 are not unexpected, given that the input sentences all contained words that resemble "shall" and a lot of the sentences could be transformed into UP. Figure 35 shows that most of the sentences in Dataset 3 were converted into UP pattern, but there are also some EDPs, SDPs, UBPs and combinations of patterns. Dataset 1, which had the worst precision and F-measure proved to have the most variation in its output patterns; only about half of them were UPs. Dataset 2 had the second smallest number of UPs in its output. Dataset 4, the literature dataset, had the largest UP percentage in its output and the worst recall of the datasets.

4.3. Ambiguity Reduction

To measure the structural ambiguity of the input and output of the EARS Converter, several measurements were chosen, since a single comprehensive measurement for it does not exist. The chosen ambiguity measures are the sentence structure, vocabulary size, readability and information retention. The sentence structure measures are the main ambiguity measurements, and the others are used to provide additional information to ensure that the loss of structural ambiguity does not make the text less readable and that it retains information and relevant terminology.

The sentence structure is assessed using the sentence tree form. Various measures are calculated for each dataset's input and output sentence trees to assess the change in structural complexity, which is tightly related to structural ambiguity. No humans were used to provide subjective assessments of the structural ambiguity because of the time and resource constraints.

The chosen measurements may be also used to assist measuring some other types of ambiguity, not only structural ambiguity. For example, syntactic ambiguity might be measured through assessing the structures of the sentence with the help of parse trees. Some lexical ambiguity could be measured indirectly through assessing the number of different words and their frequencies. Measuring semantic ambiguity might require implementing and applying semantic networks or some other way to calculate possible meanings for each word, which is out of scope of this thesis. [44] However, the frequencies and numbers of words can tell about the difficulty level of interpreting the text correctly, even if not exactly about in how many ways the text can be interpreted.

In the following subsections are described the application and results of each applied ambiguity measure. For clarity, most of the results that are in graphical form in this chapter can also be found in Appendix A in simple table form.

4.3.1. Sentence Structure

Removing structural ambiguity requires making the structures of the sentences simpler. In English, the very basic structure of sentence is typically subject-verb-object. In an example sentence from Dataset 1, "The players shall handle the chess clock properly.", "The players" is the subject, "shall handle" is the verb and "the chess clock" is the object.

The complexity of a sentence can be seen from a sentence tree. The more complicated a sentence is, the more potential it has for ambiguity.

Stanford parser was used to create parse trees to analyse sentence complicatedness. As the measures of complicatedness for each sentence was calculated the noun phrase chunk (NPC) counts, NPC to verb phrase chunk (VPC) ratios, branch counts and sentence lengths.

Sentence Length

The longer the sentences, the more potential they have for complexity and ambiguity. The lengths of sentences were calculated as the number of words (tokens) in sentences. The analysis of the sentence lengths of each dataset are visualised in Figure 36.

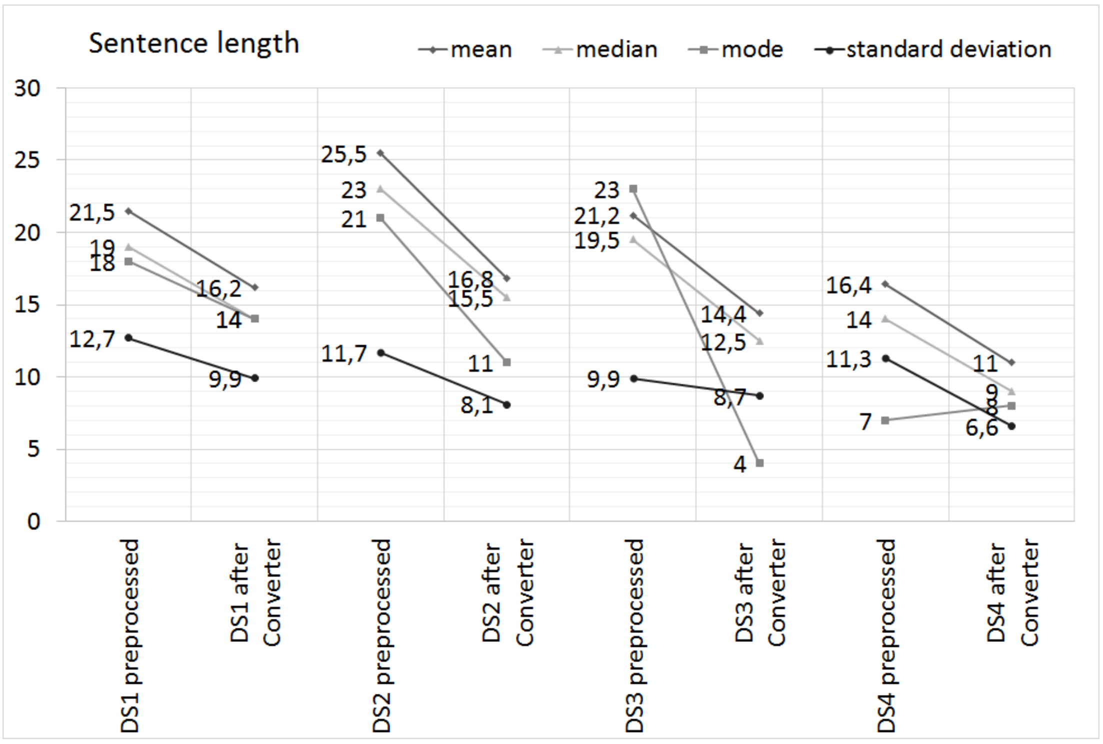


Figure 36. Analysis of the sentence lengths of each dataset after preprocessing and after the Converter.

Dataset 2 contained the longest sentences on average and Dataset 4 the shortest. The most dramatic change the converter induced in sentence lengths happened for the mode of Dataset 3: It dropped from 23 to 4. If a sentence is in EARS form, at its shortest, it can have four words (e.g. "the actor shall act", UP). The only increase was for the mode of Dataset 4 and it was minor: The mode went from 7 to 8. The standard deviations of sentence lengths all decreased, which indicates that the lengths became more uniform.

Noun Phrase Chunks

Noun Phrase Chunks (NPC) are NPs that can not be broken down to sub-NPs. NPC count tells about the complexity of a sentence. The more there are NPCs, the more complex it can be. [50]

Figure 37 shows the numbers of NPCs in a sentence for each dataset after preprocessing and after conversion. The numbers of NPCs were calculated with the help of Stanford parser. [23]

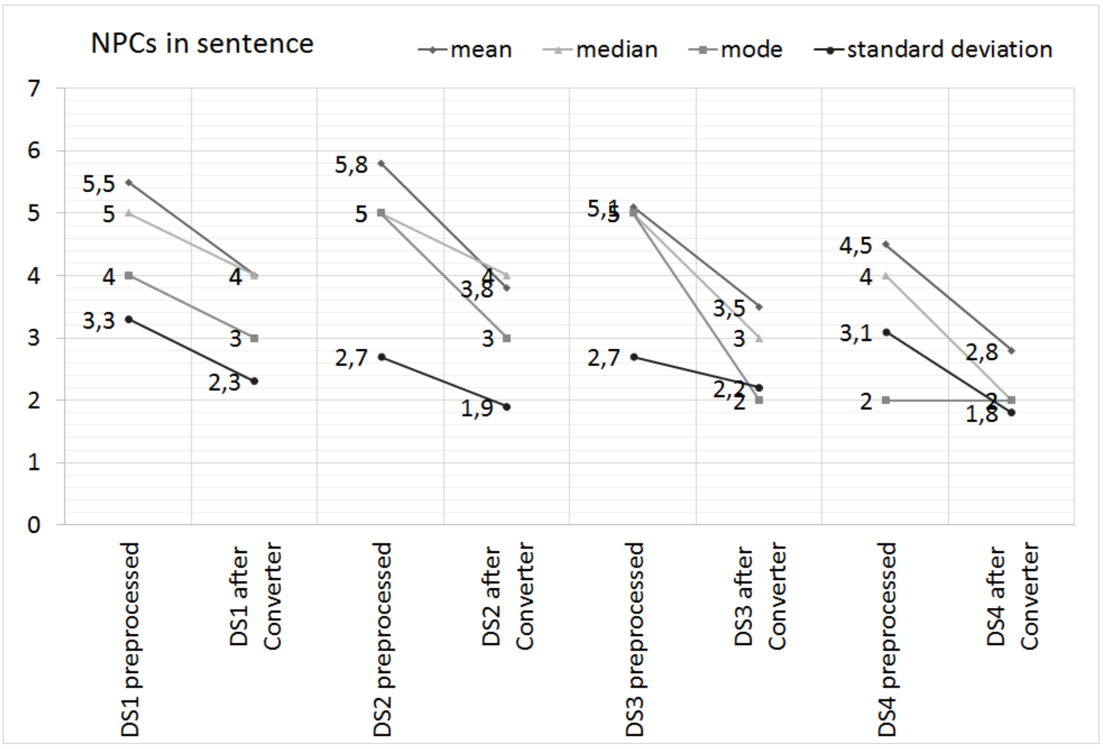


Figure 37. Analysis of the number of NPCs in sentence. The Converter appears to decrease the number of NPCs in sentence. Less NPCs in a sentence indicates that the converted sentences may be simpler than the only preprocessed sentences in the sense that they refer to a lesser amount of different entities.

The number of NPCs in sentence decreased for each dataset after conversion. The number of NPCs in sentence was originally the smallest for Dataset 4 and the largest for Dataset 2. The median NPCs in sentences was 5 for most of the preprocessed datasets. For datasets 1 and 2 the median was very similar afterwards (about 4) and datasets 3 and 4 experienced a more dramatic decrease during conversion.

NPCs to VPCs in Sentence

If a sentence contains a noun phrase (NPC) and a verb phrase (VPC), it is possible for the sentence to include an action and the actor who does the action. The NPC and VPC counts were compared against each other for each dataset before and after applying the EARS Converter.

Figure 38 shows a breakdown of the ratios of NPCs against VPCs in a sentence for each dataset after preprocessing and after conversion.

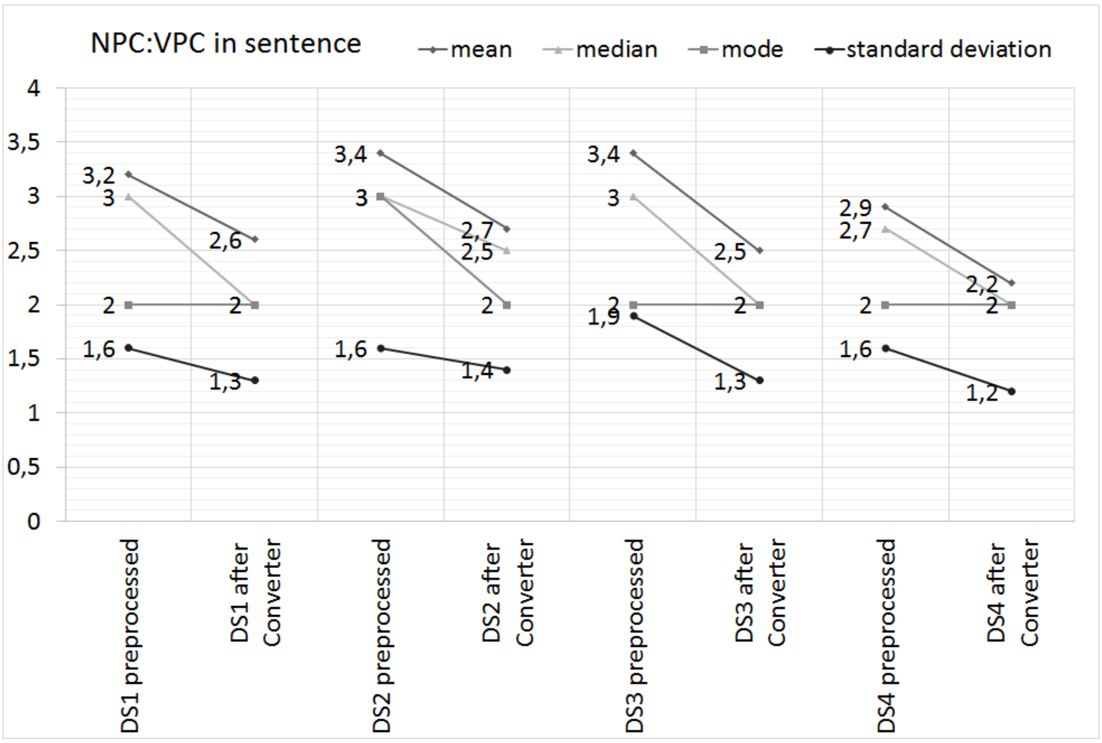


Figure 38. Analysis of the number of how many NPCs there were for each VPC in sentence. Sentences that were missing either NPCs or VPCs were omitted from the data.

The number of NPCs to VPCs decreased overall, which indicates that the sentences became simpler. The standard deviation suggests that there were less variation between the NPC to VPC ratios. Dataset 2 had the largest number of NPCs to VPCs and Dataset 4 the smallest.

The mode was the same (2 NPCs for each VPC) for each dataset after conversion. It had also been the same for almost every dataset in the preprocessed data. The sentence structure where subject follows a verb which follows the object is very common. [16] The UP typically follows the subject-verb-object sentence structure.

Branch count

The number of branches in a sentence tree describes the complicatedness of a sentence structure. In Figure 39 are depicted the mean, median, standard deviation and mode of each dataset's branch count.

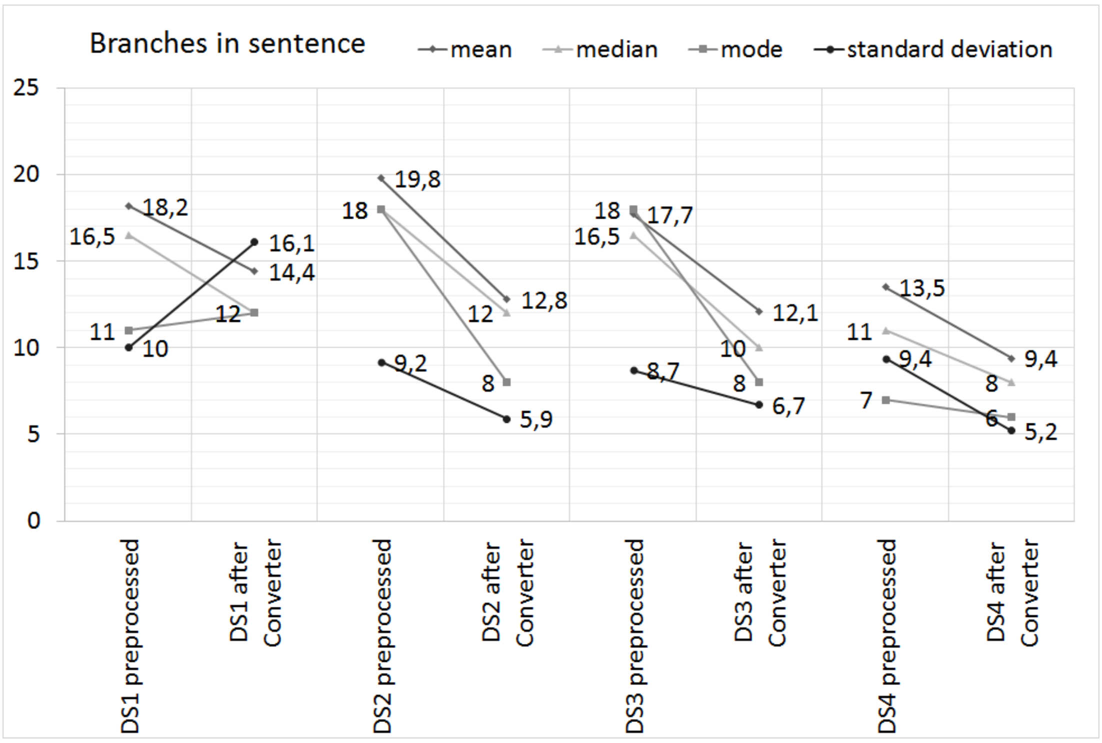


Figure 39. Branches in sentences.

All of the measures had decreased after conversion, except for Dataset 1. Dataset 1's standard deviation and mode increased after conversion, but its mean and median decreased. The explanation for that could be that the Converter omits branches from shorter sentences without omitting as many of the branches of longer sentences. For other datasets the branches are filtered more equally from shorter and longer sentences.

4.3.2. Word Occurrences

The number of occurrences of different words tells about complexity, since each different word represents a possibility of an increase in possible interpretations. If certain words appear many times in a text, it can indicate that either the word has multiple meanings or the text's topic is tightly tied to the word. Typically, word frequency distribution over different words follows the Zipf's word law [78, 79].

The vocabulary size of each dataset (before and after applying the Converter) was calculated and the word frequency distributions checked for abnormalities. This helps with assessing the changes in topic of the dataset and if the EARS structure skews the word distribution.

Vocabulary

A larger vocabulary can make a text more precise, if the vocabulary includes more than just the most common words. The most common words have more possible interpretations than less common words. [55, 56] To get a general measure of how large part of the vocabularies of the datasets are common words and possible jargon or otherwise special words, the vocabularies of the datasets were extracted and compared against a list of 3000 most common words in English language. [80]

The Oxford Text Checker [80]compares texts against the Oxford 3000^{TM} word list that includes the 3000 most common English words. It can also be used to check the texts against the 2000 most common words of the 3000 most common words and to calculate the percentage of how many words in the text appear in the Academic Word List (developed by Averil Coxhead of University of Victoria). [80] Figure 40 summarises the results of the Oxford Text Checker over the preprocessed and converted datasets.

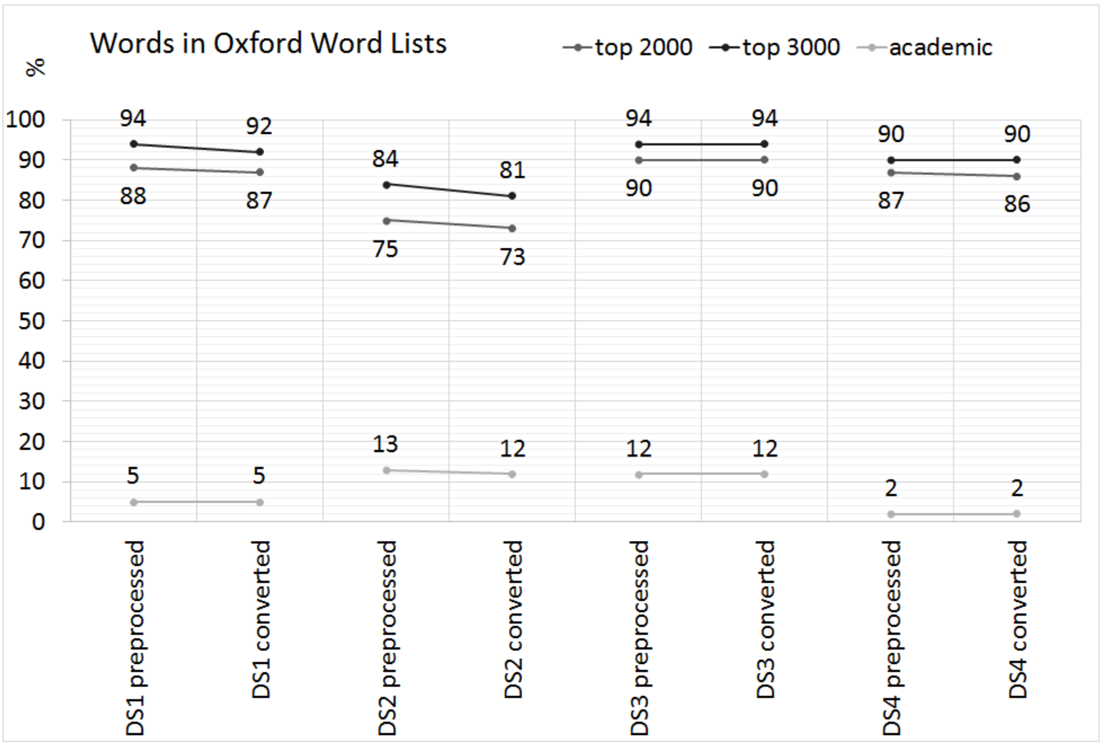


Figure 40. The percentages of words that appear in top 2000 and top 3000 most common words and the academic word list for each dataset after preprocessing and after conversion.

Word frequencies

The text of each dataset should follow the Zipf's law, like language typically does. The Zipf's law was tested on the datasets and the diverging words were analysed. The most common words of each dataset were also compared to each other, with the expectation that after the EARS Converter, they would include the words in the EARS pattern structures.

Appendix B contains Zipf plots for each dataset before and after the conversion. Appendix C contains the twenty most frequent words of each dataset after preprocessing and after conversion. From the Zipf plots can be seen how the word count distribution approximately follows the Zipf curve, as expected.

The most common words in all of the converted datasets are "the" and "shall". Those are the marker words of UP, and they appear in all of the EARS patterns. The third most common word in the converted datasets is "be" in all the datasets except Dataset 3, where it is "program".

In the preprocessed Dataset 1 the second most common word ("a") is unexpectedly rare compared to the most common word ("the"). The distribution evens out after applying the Converter, as the second most common word changes to "shall".

Datasets 1, 2 and 3 contain marker words for EARS sentences, stopwords and other relevant words in the 20 most common words.

In Dataset 1, "player", "move", "game" and "arbiter" appear in the top 20 list before and after conversion. Those words summarise well the topic of the dataset, which is rules for a game.

In Dataset 2, "must", "ip", "host", "address" and "tcp" appear in the top 20 words both before and after conversion. The word "may" disappears from the top 20 list after conversion. The Converter transforms "may", in a correct position, into "should", so this is expected. The word "layer" appears in the top 20 of Dataset 2 after conversion. The Converter does not add that word to sentences, but it does transform words like "may" and "should" into "shall" in some cases, which may be the reason for its rise on the list.

Dataset 3's top 20 words contain the most non-stop words of the datasets. The preprocessed dataset's notable non-stop words are "program", "message", "physical", "units", "mode", "unit", "water", "sent" and "failure". In conversion "unit" drops out of the top 20 list and "stop" and "level" rise on it.

Dataset 4 is the largest of the datasets and it contains the most general text. The top 20 words of it only contain stop words and after conversion stop words and EARS pattern words. The "os", which appears as the 17th most common word in the converted Dataset 4 is a word that is added by the Converter when it does not find an actor candidate.

4.3.3. Readability

To measure reading comprehension, the Automated Readability Index (ARI) was calculated for each dataset before and after the Converter. ARI tells an estimation of on which (United States) school grade level the reader should understand the text. [57]

In Table 7, ARI has been calculated for each dataset before preprocessing. This was done so that the preprocessed and converted datasets could be compared against the original texts, which are quite different from even the preprocessed datasets. The strokes are visible (non-whitespace) characters, words are lines of characters separated by whitespace characters and sentences are sentences extracted using NLTK corpus reader. The score is calculated with the ARI formula [57] in Equation 6 (Chapter 2 Section 2.2.4.

Table 7. ARI before preprocessing

	strokes	words	sentences	score
Dataset 1	50 976	9 233	616	12.1
Dataset 2	226 777	34 759	4 929	12.8
Dataset 3	11 603	1 887	95	17.5
Dataset 4	672 648	121 850	15 274	8.6

In Table 8 the ARI has been calculated for the datasets after preprocessing. After preprocessing, each line had the name of the requirement it represents as its first word. It was removed for calculating the ARI. The level of difficulty of reading comprehension rose for all datasets except for Dataset 3. Datasets 1, 2 and 3 lost data in preprocessing, since only certain parts were chosen as suitable for conversion as potential requirements.

Table 8. ARI after preprocessing

	strokes	words	sentences	score
Dataset 1	27 942	5 056	240	15.1
Dataset 2	57 988	9 626	397	19.1
Dataset 3	10 560	1 780	86	16.9
Dataset 4	676 123	121 476	7 657	12.7

Table 9 shows the ARI for datasets for the final result after conversion. The Converter decreased ARI for every dataset. All the other fields of the table also decreased or stayed the same. Datasets 1, 2 and 4 lost sentences because the Converter interprets some sentences as unconvertable.

Table 9. ARI after conversion

	strokes	words	sentences	score
Dataset 1	20 998	3 890	234	12.3
Dataset 2	39 018	6 629	393	14.7
Dataset 3	7 276	1 241	86	13.4
Dataset 4	415 119	77 885	7 047	9.2

Datasets 2 and 3 are the most difficult ones to read in their original form and also in their converted form. This is not surprising, because both of them consist of technical specification text. However, the level of difficulty decreases more for Dataset 2 than Dataset 3. This may be because the style of Dataset 2 is originally in a less technical format and it becomes more so. Dataset 3, on the other hand, is originally in a format that does not need to change as much to be in EARS form.

Dataset 1 describes a system that involves rules for humans rather than machines. The text is meant for more casual readers than datasets 2 and 3 that are meant for technical people. That may explain why its ARI is lower. Dataset 4 is meant to read for leisure, which explains why it is the easiest dataset to read.

4.3.4. Information Retention

Change in information complexity was used as the measure for information retention. The diversity of data and how tightly it can be packed tells about the information complexity. Calculating the Kolmogorov complexity for each dataset is used for comparing the differences in information complexity of the datasets before and after conversion. [46] If the information complexity decreases, that means the data may become less meaningful, since the percentage of information it contains decreases.

The Kolmogorov complexity tells the potential for complexity from information point of view rather than on word or sentence level. With the help of file compression programs, finding out Kolmogorov complexity is relatively easy. The dataset is stored in a file and the file is compressed using a lossless compression algorithm. Then the file sizes after and before compressing are compared against each other.

The chosen compression algorithm was 7zip's LZMA2 due to its free availability. LZMA2 is superior to the LZMA algorithm, which is its predecessor. [47] The LZMA (and LZMA2) uses several compression techniques one after another to achieve the resulting compression. The compression of LZMA is bitwise rather than bytewise, which makes it better than many of its competitors. [81] Table 10 shows the file sizes before and after compression.

Table 10. File sizes and compression ratios

		Original (bytes)	Compressed (bytes)	ratio (%)
Dataset 1	preprocessed	28 797	8 218	28.538
	converted	23 119	6 516	28.185
Dataset 2	preprocessed	58 852	17 361	29.499
	converted	43,368	12 154	28.025
Dataset 3	preprocessed	11 285	2 657	23.545
	converted	8 897	2 070	23.266
Dataset 4	preprocessed	695 512	199 147	28.633
	converted	476 047	124 266	26.104

Figure 41 illustrates the differences between compression ratios of each dataset before and after conversion. Each dataset's compression ratio decreases in conversion, but not significantly. They stay about the same.

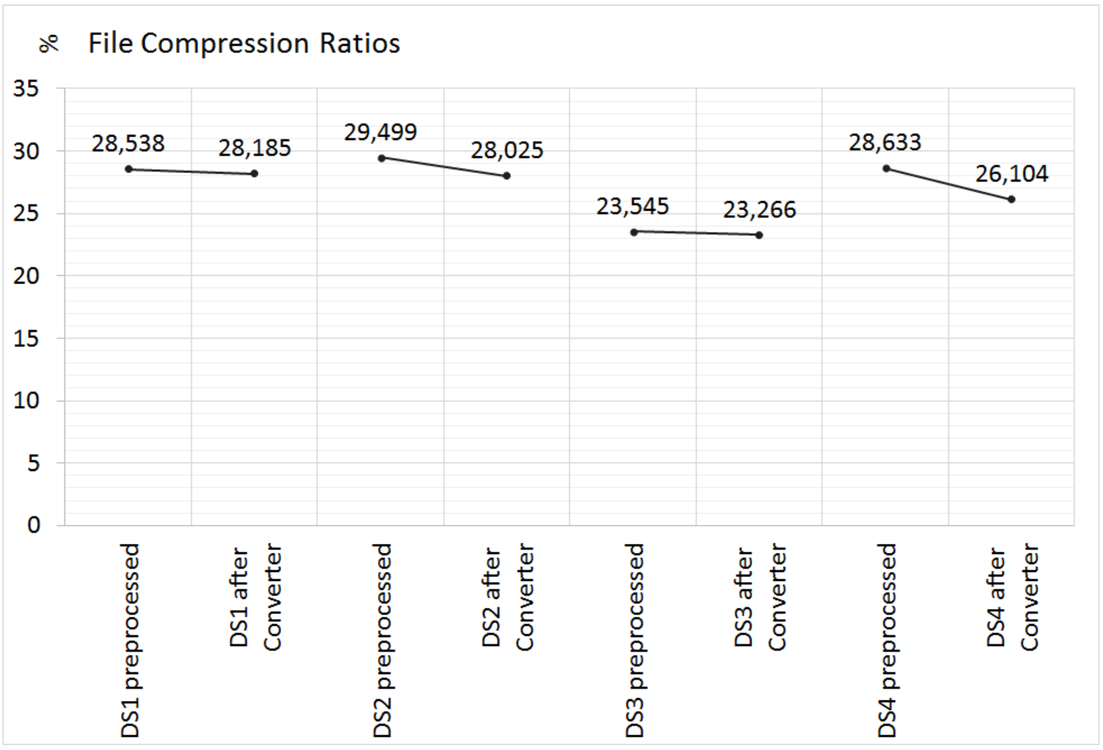


Figure 41. The compression ratios of each dataset after preprocessing and after conversion.

4.4. Summary of results

The Converter converted almost all of its input sentences into EARS if they were not in it already. It also proved it could transform sentences into all of the different EARS pattern forms. However, it is inconclusive whether the new sentences make sense, since qualitative assessment of the sentences could not be done due to the time and resource constraints. It is recommended that the EARS Converter is used by a requirements engineer to check the resulting sentences, rather than blindly transforming sentences and leaving them in the transformed form without checking.

The conversion reduced structural ambiguity. The sentence lengths decreased, which decreases the potential for complexity of sentences. The sentences contained less NPCs and the NPC to VPC ratio decreased, demonstrating the simplification of the structure. The readability index indicates that the sentences became more readable after conversion.

The performance results were rather good. The recall and precision of each dataset were all above 86% for all datasets and F-measure was above 92%. For one dataset they all were 100%. The Converter managed to transform NL sentences into EARS.

According to the used measures, it appears that the Converter can reduce structural ambiguity. The structures of the sentences became simpler: The NPC to VPC ratio got better, there were less NPCs and branches overall in a sentence and the sentences became shorter. The number of word occurrences distribution of the datasets followed Zipf's law before and after conversion. The readability scores after conversion improved. Information complexity decreased only a little, which means that the transformed sentences are about as informationally rich as the original sentences.

5. DISCUSSION

In this thesis was presented a tool (the EARS Converter) for turning natural language (NL) software requirements specification (SRS) sentences into EARS form. The Converter was tested on four datasets and it reduced structural ambiguity in all of them.

The applied testing methods for ambiguity reduction were quantitative. Had there been more time and resources, more qualitative testing methods would have been applied to assess the overall ambiguity better. With the quantitative measurements only structural ambiguity was measured.

Unlike with many other studies on SRS, the datasets used for this thesis are freely available on the Internet. Thus, the results of this thesis can be used as a point of comparison for future structural ambiguity reduction systems.

Since the EARS Converter is in its proof-of-concept form, there are ways that it could be improved. With further development, the Converter could be combined with, for example, a system like ARSENAL [82] that transforms EARS-like sentences into formal form.

5.1. Results

The results suggest that the Converter reduces structural ambiguity. To remove other types of ambiguity and to measure their removal, more studies are required.

During the development of the Converter, some SRS specific problems with the Stanford parser were found that may apply to other parsers as well. For example, the parser misinterprets sentences more easily if some of the words are in all uppercase letters. That particular problem was fixed in preprocessing by turning the critical words (e.g. MUST and MAY) to their lowercase counterparts if they were written in uppercase. Unfortunately, it is likely that some other similar problems went unnoticed during the development of the EARS converter. These kinds of problems arise, because the parser was not trained on SRS data.

The preprocessed input and output of the EARS Converter were compared against each other regarding their structural ambiguity. The chosen measures in this thesis were mostly statistical to achieve the largest possible level of automation and objectivity. However, when measuring ambiguity, subjective interpretations are relevant. Given more time and resources, subjective ambiguity measures would have been also used.

In some cases complexity and ambiguity are words that can be used interchangeably, but they actually represent different concepts. Complexity is related to how many parts and connections a thing has, and ambiguity is related to how the thing is understood. In this thesis information complexity was measured as an aid to measuring the change in ambiguity.

5.1.1. Performance

The datasets contained mostly sentences with a main verb, which is the bare minimum for the Converter to be able to transform a sentence into EARS form. Thus, the

Converter transformed almost all input sentences into EARS form. However, it must be noted that EARS form is not a guarantee that the sentence makes sense, and with more time and resources that would have been measured also with the help of human evaluators to assess that.

To the best of the authors knowledge, other equivalents of the EARS Converter have not been built, which means that there are no competitors to compare to. One example of a somewhat similar system is PROPEL ("Property Elucidation")[83], which analyses NL and elucidates the properties of the requirements [84] instead of converting the sentences to EARS form.

5.1.2. Ambiguity Reduction

This thesis focused on structural ambiguity only. Since the types of ambiguity are not easily separable from each other, measuring structural ambiguity involves measuring other types of ambiguity as well.

Lexical ambiguity was lightly evaluated to get a better sense of the changes in structural ambiguity. The number of occurrences of each word, especially of the most common words, tells a lot about the readability of the texts. If structural ambiguity decreases, also its readability should get better and it can also affect the numbers of words and word complexity.

Structure

Analysing the sentence structure tells the most about structural ambiguity. In this thesis, noun phrase chunks (NPC) and verb phrase chunks (VPC), as well as the number of branches in sentence trees and sentence lengths were calculated for the analysis. With these, some simple analyses could be performed to determine the levels of ambiguity of sentence structures.

Information Retention

Information retention was measured to ensure that the Converter does not just translate the sentences into pure structure without content. The measuring was done by applying Kolmogorov complexity: the compression ratios of files before and after the Converter were calculated.

As the results, the compression ratios did not change much, but the compression was a little bit more effective after conversion. This means that in the restructuring process a little bit of the information density was lost.

In addition to the density of data, some of the quantity of the data was also lost. The Converter prunes the sentence trees as it creates new sentences. Some of the pruned branches may have contained important side notes, but that can not be measured merely by comparing the information density and amount of text.

Readability

According to the readability scores based on word complexity, the Converter made the texts easier to read. Automated Readability Index (ARI)[57] was applied to measure the readability in this thesis. It measures the readability based on the text only by the characters and words, not by interpretations or structures. It could be argued that it is a very surface level readability index, but ARI and other readability indices, are often objective. They assess the text by the numbers, not the meaning.

The target reader group of SRS documents is software engineers and software customer representatives. To measure how well the target group understands a text, a group of human evaluators representing the target group may be required. They should also read the documents and respond to questions about it.

5.2. Future Work

The EARS Converter has potential uses in different systems. The solution of this thesis, if developed further, could be used with tools like traceability matrices [85] to help managing the changes in requirements. Another potential use for a further developed EARS Converter would be a system that automatically produces code when a NL SRS is given to it as an input.

5.2.1. Further Development of the Converter

There are many ways the Converter could be developed further. The output quality could be improved by stemming the words to be transformed [1] and the pronouns could be mapped to the likely previous nouns they are referring [2].

The Converter at its current state only reduces structural ambiguity, but it could also be expanded to reduce lexical ambiguity.

Stemming

Stemming in general helps with connecting words with their roots. That way it helps with connecting them to more of their possible meanings. With the help of tools like a glossary or semantic networks, stemming could help with reducing lexical ambiguity. In the current implementation stemming is not utilised much – only a little bit with turning verbs into more correct forms when transforming the sentences in the Ubiquitous pattern form.

In the current implementation '-s' or '-ed' is removed from the end of the verb when present to make the verb more correct, but it can not track other kinds of past tense

[1]The Converter only stems and transforms some verbs to fit better with the word "should" at this point.

[2]Currently the Converter only adds the words "The OS" to potential UP sentences that do not have a subject.

to present tense transformations than just removing '-ed'. Verbs 'is', 'are', 'was' and 'were' are transformed in 'be'-form in the same manner.

Stop Word Removal

In natural language, stop words are frequently appearing words with little meaning. For example, "the", "and" and "when" are words that can be seen as stop words. There are different ways to determine which words are stop words, depending on the context. Thus, different lists of stop words exist.

Stop word removal is one of the key techniques in NLP. Reducing less useful words can make extracting information more efficient when dealing with large pools of data and trying to find a general consensus about a specific thing. Key words can mix up with stop words, which also can make it useful to remove stop words before trying to find key words.

The structure of EARS patterns consists of words that appear frequently, especially in text written in EARS. The structural words of a sentence may have only little meaning, but they provide nuances that can be essential for writing and deciphering requirements.

Stop words were not removed from the input data, since that would remove too many of the possible key words.

Automated glossary creation

If the vocabulary with the meanings of the words of an SRS could be reliably extracted automatically, dispersing lexical ambiguity would become easier. Extracting the vocabulary is relatively easy by extracting all the words, but extracting the meanings of words and phrases and the connections between them is not as simple. For connecting the meaning with the words, semantic networks could be used.

Semantic networks could help with reducing semantic or lexical ambiguity by transforming sentences using the possible interpretations of the words to find alternative words. Semantic networks could also be used for finding whether different sentences are talking about same things. They could help with keeping track of requirement changes in the SRS and with mapping which requirements are related to which aspects of the software. Systems like PROPEL[83] could also benefit from SRS specific semantic networks.

The existing semantic networks, like the Semantic Web [86], have not been created specifically about SRS vocabulary. Despite their more general nature, some semantic networks could still be used on SRS documents to reduce at least some lexical ambiguity. However, that requires more research.

5.2.2. Ambiguity Measures

Transforming sentences in their EARS equivalents does not remove the ambiguity of the nouns and verbs and subclauses involved. The expressions themselves can be ambiguous or the things they refer to can also be ambiguous.

The ambiguity measures used in this thesis were chosen for structural ambiguity. Other types of ambiguity measures, like subjective assessments, may be needed for measuring other types of ambiguity. [44]

5.2.3. Feature Extraction

With feature extraction a lot could be learned from SRS documents. For example, different types of SRS documents could be recognised before processing and, if combined with the Converter, the preprocessing could be done more automatically. Some feature extraction approaches have been tried before [87] but there is a lot of room to learn more.

Keyword collection is one type of feature extraction. A very modest amount of keyword collection was applied in the testing and analysis of the results of this thesis. It was done to give a bit of perspective to the datasets and what they are about.

Keyword collection could be used to a much larger extent in analysing SRS documents. It could give an initial view on the topic of the SRS and what may be its most important words. With a glossary to map words to each other, there could be ways to find what words are synonymous to the most relevant words and replace the synonyms with the most relevant words. This would be useful, because when freely writing NL, people do not always use the same words when they mean the same thing.

If automated keyword extraction was meant for finding structures instead of words, it could have been helpful for the Converter. However, even more advanced keyword extraction tools like RAKE [31], remove stop words before extracting keywords. Thus keyword extraction would be difficult to use for finding structures.

5.2.4. NLP and Requirements Engineering

Requirements engineering relies on requirements specifications (RS), which are typically in the form of text documents. At least in software engineering, the RS typically contains a lot of NL. When NL is present, NLP techniques can be useful. [88]

NLP can be used to analyse documents, build better documents and to keep track of changes. With machine translation, they can even be translated to other languages. Making better use of NLP could be beneficial for software requirements engineering.

CNL linguistic studies on EARS

SRS documents can apply controlled natural languages (CNL) in addition to NLs. A lot of research has been done to build different CNLs, but more could be made. Now that NLP has progressed to its current level, machine translation with CNLs should be studied more.

SRS benchmarks

One of the largest hindrances for the study of NLP with requirements engineering is the lack of large freely available SRS databases. Since a good benchmark database does not exist, comparing studies is not easy. Publishing studies concerning SRS documents can also be difficult if the documents themselves have to be kept secret.

There are some possible solutions for this problem, but all of them require a lot of time, resources, effort, and management. For example, group of companies could together provide a pool of SRS documents for everyone to peruse. That would require arranging such an agreement between the companies. It could be difficult to get a large enough number of companies to agree about letting such well guarded documents go. For another example, open source community could start building SRS documents for open source tools, but producing SRS documents requires effort that people working for free may not have the incentive to do. Such efforts might not be feasible.

The nlrpBENCH [70] is a good example of a project that attempts to fix the lack of freely available SRS data to study. It was applied in this thesis for finding suitable data for testing. Unfortunately, as good as the idea behind it is, it does not include all kinds of SRS data and no annotated EARS data or other types of annotated data. Further gathering efforts of a freely available SRS databases would be welcome for NLP studies concerning SRS documents.

Until a large and diverse SRS database is available [3], SRS studies need to come up with different ways to cope with the SRS data shortage. In this thesis the four chosen datasets were small, but they represented different forms of the text that SRS documents may contain. Other ways to cope with the difficulty of finding suitable data could include making the studies private so that companies do not have to publish their SRS documents. That, however, is not very desirable from the scientific point of view.

Turning NL SRS Into Different Forms

NL SRS documents are a blessing and a curse, since NL is so versatile and easy to understand, but also ambiguous. The idea behind EARS Converter arose from the hope that someday it would be possible to convert NL SRS into executable code. The Converter is the first step towards that goal.

NL has already been turned into some formal forms successfully.[89] For example, it has been attempted with neural attention. [37] Parts of NL have also been translated into description logic [9] and CNL has been used to express logical arguments [3]. However, turning language to logic or other formal forms usually requires that the language has enough structure. [26] The EARS Converter could be used as a part of such system to perform initial transformation of the language into more structured form.

[3]If one ever is! Businesses like to keep their secrets.

6. CONCLUSION

This is a master's thesis in computer science and engineering at the University of Oulu. The work was done for a private company in 2018. The goal was to produce a tool for reducing ambiguity of SRS documents. The goal was met. The EARS Converter proved to be able to transform sentences into less structurally ambiguous form.

In addition to the EARS Converter, for this thesis was produced the EARS Recogniser to automatically recognise EARS sentence structures. This thesis also provides a set of measures to assess structural ambiguity, and a group of datasets that can be used as a small benchmark for further studies of ambiguity reduction in SRS documents.

As it was noted in the Introduction chapter, ambiguity of SRS can cause a lot of expenses. Applying a system like the EARS Converter does not add much to the expenses. Instead, it can be used to reduce the ambiguity of SRS from the beginning of writing the SRS. The earlier the ambiguity is resolved, the less issues it is possible for it to cause.

Ambiguity is a challenging thing to grasp. [39] People can understand the same sentence in many different nuanced ways. Reducing structural ambiguity means reducing the number of ways a sentence can be interpreted according to its structure. In this thesis, the EARS Converter was applied to restructure NL sentences to EARS to reduce structural ambiguity. There is still a lot of work to do to find ways to reduce other types of ambiguity.

The EARS Converter can be applied without having to give a lot of training to its users. The Converter simply outputs the EARS form interpretations of the input NL sentences. The user does not even need to understand EARS format to use it. The output sentences can be used as a metric for whether or not the Converter understands the sentence structure or as suggestions for EARS form sentences.

The results suggest that the EARS form sentences are less structurally ambiguous than the NL input sentences. The tree form of the sentences was used to analyse the ambiguity of sentence structure, a bit like how chunks have been previously used for such tasks [50]. The tree form measures were supplemented by some more general measures, like the sentence lengths, vocabulary variation and readability index.

The sentence structure can be gathered from the sentence's sentence tree form. In automatic analysis of the sentence, the builders of the sentence trees are parsers, like the Stanford parser which was used in this thesis. [23, 30] Humans can also construct sentence trees, but automatic creation of them is faster and it can also be more accurate. [71] However, even the Stanford parser, which is one of the most advanced parsers available, did not accurately parse sentences when they were missing periods or some of the words were written in all uppercase letters. More research on parsers with jargon-rich data, like the SRS documents, is recommended.

The EARS Converter can be used as the first step towards a system that automatically turns NL SRS into different formal forms. However, the Converter currently only reduces structural ambiguity. Other types of ambiguity have to be removed from sentences before the sentences are ready for reliable unsupervised transformations.

Previous work on natural language processing (NLP) with requirements engineering (RE) has mostly focused on sentences that have already been in CNL form. [1, 6] In contrast, the solution in this thesis transforms NL into CNL. This facilitates building systems that might be able to automatically turn NL into code or other formal forms.

This would be useful in requirement management systems, since it might make it possible to translate the requirements into executable code.

This thesis presented a point of measure for turning NL sentences into EARS. The datasets are also freely available, which helps with the problem of not having enough freely available benchmarks. [63, 70] Precision, recall and F-measure were used to assess the performance of the Converter. Despite not being the best for every situation [33], these measures are common in NLP and likely to be familiar to many NLP researchers.

Perfectly objective ambiguity measures do not exist. [38, 44] This thesis applied a group of structural ambiguity measures that at least aim towards objectivity. All of them are quantitative. Further studies are required, however, to determine whether they are enough for all kinds of datasets and what else can be used.

With further development, more ambiguity types could be reduced with the EARS Converter. An example of improving it is including automated glossary creation, which could be used to find connections between requirements. With the development of glossary creation and applying it, the Converter could be become an alternative to systems like ARSENAL [82]. Another example would be simply to improve the current version of the Converter by stemming words, or manually adding more types of possible condition transformations. One more example of a possible direction of development is building a module that compares the requirements sentences against each other as a whole, or by their structure.

The Converter can be seen as a simple natural language understanding (NLU) system. It interprets NL in another form. More complicated NLU systems could be built that would use such interpretations as, for example, a base for better understanding of the software. If an NLU system can analyse the requirements, it might be possible for it to check whether the requirements and the software match. That could enable more automated software testing and development.

More research that join software RE and NLU could lead to higher quality automatic software production. However, the insights from the RE-NLU studies could also be used in other areas of science as well, since NLU requires more than just NLP. For example, in addition to NLP, the results of this thesis unite ambiguity studies, linguistics and RE. Perhaps some day it becomes possible for a machine to understand what sort of code humans want it to produce automatically.

6.1. Perspectives

There were some interesting limitations during creating the Converter and choosing the measures. The parser would probably have performed better with more specific training data. The types of ambiguity to reduce had to be limited to only one because ambiguity is an extensive concept.

The Stanford Parser was not trained on the type of data SRS typically contains, which is very specific technical language, even if it is often NL. The vocabulary consists of words that have very specific meanings within the context and without training the parser on a labelled specific dataset, the results may not be as specific as they could be. To have such datasets, perhaps creating an AI using deep learning to label technical

NL data would help, since that could make it possible to use the company's own data to train the parser without having to release any company data outside.

Words and sentences can be interpreted in many different ways depending on the context. In technical documents, the context is relatively simple compared to, for example, in novels. Even in technical documents, context is hard to grasp tightly enough to force only one possible interpretation on one sentence. Programming languages are much less ambiguous in this regard, since their context is very limited. Bringing sentences even closer to programming languages than EARS patterns are could further reduce their ambiguity through limiting the context. However, that would require finding or coming up with a good enough restricted language, which is a worthy task by itself.

If people are to be made use more restricted languages in requirements engineering, studies on how to make people use the restricted languages most efficiently should be made. For example, it should be verified whether it would be more efficient to have people use something like the EARS Converter to transform their NL sentences or properly teach the people to use a restricted language, like EARS, by themselves.

Ambiguity is a fascinating concept and there are many more ways to study it aided by technology. With the aid of different parsers sentences can be analysed further and faster than before. Large databases of various meanings in various contexts can be created, even if it requires the help of actual humans in labelling the data at the beginning. Catching all the possible meanings might be impossible because of the evolving nature of NL, but adding context to text might limit them enough to approach human level understanding of sentences.

7. REFERENCES

[1] Shah U.S. & Jinwala D.C. (2015) Resolving ambiguities in natural language software requirements: a comprehensive survey. ACM SIGSOFT Software Engineering Notes 40, pp. 1–7.

[2] Kuhn T. (2014) A survey and classification of controlled natural languages. Computational Linguistics 40, pp. 121–170.

[3] Strass H. & Wyner A. (2017) On automated defeasible reasoning with controlled natural language and argumentation.

[4] Kiyavitskaya N., Zeni N., Mich L. & Berry D.M. (2008) Requirements for tools for ambiguity identification and measurement in natural language requirements specifications. Requirements engineering 13, pp. 207–239.

[5] Arora C. (2016) Automated analysis of natural-language requirements using natural language processing. Ph.D. thesis, University of Luxembourg, Luxembourg.

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

[7] Xu X., Liu C. & Song D. (2017) Sqlnet: Generating structured queries from natural language without reinforcement learning. CoRR abs/1711.04436. URL: `http://arxiv.org/abs/1711.04436`.

[8] Zhong V., Xiong C. & Socher R. (2017) Seq2sql: Generating structured queries from natural language using reinforcement learning. CoRR abs/1709.00103. URL: `http://arxiv.org/abs/1709.00103`.

[9] Gyawali B., Shimorina A., Gardent C., Cruz-Lara S. & Mahfoudh M. (2017) Mapping natural language to description logic. In: E. Blomqvist, D. Maynard, A. Gangemi, R. Hoekstra, P. Hitzler & O. Hartig (eds.) The Semantic Web, Springer International Publishing, Cham, pp. 273–288.

[10] Mavin A., Wilkinson P., Harwood A. & Novak M. (2009) Easy approach to requirements syntax (ears). In: Requirements Engineering Conference, 2009. RE'09. 17th IEEE International, IEEE, pp. 317–322.

[11] Schuster S. & Manning C.D. (2016) Enhanced english universal dependencies: An improved representation for natural language understanding tasks. LREC.

[12] Osborne M. & MacNish C. (1996) Processing natural language software requirement specifications. In: Requirements Engineering, 1996., Proceedings of the Second International Conference on, IEEE, pp. 229–236.

[13] Cambria E. & White B. (2014) Jumping nlp curves: A review of natural language processing research. IEEE Computational intelligence magazine 9, pp. 48–57.

[14] Greenberg J.H. (1963) Some universals of grammar with particular reference to the order of meaningful elements. Universals of language 2, pp. 73–113.

[15] Ramat P. (2009) How universal are linguistic categories? In: Universals of Language Today, Springer, pp. 1–11.

[16] Dryer M.S. (1992) The greenbergian word order correlations. Language , pp. 81–138.

[17] Council B. (2018), English grammar. `https://learnenglish.britishcouncil.org/en/english-grammar`. Accessed: 2018-06-04.

[18] Dryer M.S. (2009), The branching direction theory of word order correlations revisited.

[19] Terzakis J. (2013) Ears: The easy approach to requirements syntax. In: Tutorial at the Eighth International Multi-Conference on Computing in the Global Information Technology.

[20] Fuchs N.E. & Schwitter R. (1996) Attempto controlled english (ace). arXiv preprint cmp-lg/9603003 .

[21] Marcus M.P., Marcinkiewicz M.A. & Santorini B. (1993) Building a large annotated corpus of english: The penn treebank. Computational linguistics 19, pp. 313–330.

[22] Tian Y. & Lo D. (2015) A comparative study on the effectiveness of part-of-speech tagging techniques on bug reports. In: Software Analysis, Evolution and Reengineering (SANER), 2015 IEEE 22nd International Conference on, IEEE, pp. 570–574.

[23] Manning C., Surdeanu M., Bauer J., Finkel J., Bethard S. & McClosky D. (2014) The stanford corenlp natural language processing toolkit. In: Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations, pp. 55–60.

[24] Schmid H. (2013) Probabilistic part-of-speech tagging using decision trees. In: New methods in language processing, p. 154.

[25] Ratinov L. & Roth D. (2009) Design challenges and misconceptions in named entity recognition. In: Proceedings of the Thirteenth Conference on Computational Natural Language Learning, Association for Computational Linguistics, pp. 147–155.

[26] Zettlemoyer L.S. & Collins M. (2012) Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. arXiv preprint arXiv:1207.1420 .

[27] Kuhn R. & De Mori R. (1995) The application of semantic classification trees to natural language understanding. IEEE transactions on pattern analysis and machine intelligence 17, pp. 449–460.

[28] Socher R., Lin C.C., Manning C. & Ng A.Y. (2011) Parsing natural scenes and natural language with recursive neural networks. In: Proceedings of the 28th international conference on machine learning (ICML-11), pp. 129–136.

[29] Bird S., Klein E. & Loper E. (2009) Natural language processing with Python. " O'Reilly Media, Inc.".

[30] Perkins J. (2014) Python 3 text processing with NLTK 3 cookbook. Packt Publishing Ltd.

[31] Rose S., Engel D., Cramer N. & Cowley W. (2010) Automatic keyword extraction from individual documents. Text Mining: Applications and Theory , pp. 1–20.

[32] Cowie J. & Lehnert W. (1996) Information extraction. Communications of the ACM 39, pp. 80–91.

[33] Powers D.M. (2011) Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation .

[34] Yampolskiy R.V. (2013) Turing test as a defining feature of ai-completeness. In: Artificial intelligence, evolutionary computing and metaheuristics, Springer, pp. 3–17.

[35] Alm C.O. (2011) Subjective natural language problems: Motivations, applications, characterizations, and implications. In: Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers-Volume 2, Association for Computational Linguistics, pp. 107–112.

[36] Yi J., Nasukawa T., Bunescu R. & Niblack W. (2003) Sentiment analyzer: Extracting sentiments about a given topic using natural language processing techniques. In: Data Mining, 2003. ICDM 2003. Third IEEE International Conference on, IEEE, pp. 427–434.

[37] Dong L. & Lapata M. (2016) Language to logical form with neural attention. arXiv preprint arXiv:1601.01280 .

[38] MacKay D.G. & Bever T.G. (1967) In search of ambiguity. Perception & Psychophysics 2, pp. 193–200.

[39] Massey A.K., Rutledge R.L., Antón A.I., Hemmings J.D. & Swire P.P. (2015) A strategy for addressing ambiguity in regulatory requirements. Tech. rep., Georgia Institute of Technology.

[40] Wasow T., Perfors A. & Beaver D. (2005) The puzzle of ambiguity. Morphology and the web of grammar: Essays in memory of Steven G. Lapointe , pp. 265–282.

[41] Mesheryakov D.K. & Selegey V.P. (2018), Language ambiguity detection of text. US Patent 9,984,071.

[42] Ballesteros L. & Croft W.B. (1998) Resolving ambiguity for cross-language retrieval. In: Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval, ACM, pp. 64–71.

[43] Ratnaparkhi A. (1998) Maximum entropy models for natural language ambiguity resolution .

[44] Mich L. & Garigliano R. (2000) Ambiguity measures in requirement engineering. In: International Conference on Software Theory and Practice. ICS.

[45] Pallotti G. (2015) A simple view of linguistic complexity. Second Language Research 31, pp. 117–134.

[46] Ehret K. & Szmrecsanyi B. (2016) An information-theoretic approach to assess linguistic complexity. Complexity and isolation. Berlin: de Gruyter .

[47] Langiu A. (2013) On parsing optimality for dictionary-based text compression—the zip case. Journal of Discrete Algorithms 20, pp. 65–70.

[48] Jay G., Hale J.E., Smith R.K., Hale D.P., Kraft N.A. & Ward C. (2009) Cyclomatic complexity and lines of code: Empirical evidence of a stable linear relationship. JSEA 2, pp. 137–143.

[49] Housen A. & Kuiken F. (2009) Complexity, accuracy, and fluency in second language acquisition. Applied linguistics 30, pp. 461–473.

[50] Din C.Y. & Rine D. (2008) Requirements content goodness and complexity measurement based on NP chunks. VDM Publishing.

[51] Anderson R.C. & Pearson P.D. (1984) A schema-theoretic view of basic processes in reading comprehension. Handbook of reading research 1, pp. 255–291.

[52] Simon H.A. (1974) How big is a chunk?: By combining data from several experiments, a basic human memory unit can be identified and measured. Science 183, pp. 482–488.

[53] Snow C. (2002) Reading for understanding: Toward an R&D program in reading comprehension. Rand Corporation, 19–28 p.

[54] Hock M. & Mellard D. (2005) Reading comprehension strategies for adult literacy outcomes. Journal of Adolescent & Adult Literacy 49, pp. 192–200.

[55] Laufer B. & Ravenhorst-Kalovski G.C. (2010) Lexical threshold revisited: Lexical text coverage, learners' vocabulary size and reading comprehension. Reading in a foreign language 22, pp. 15–30.

[56] Hirsh D., Nation P. et al. (1992) What vocabulary size is needed to read unsimplified texts for pleasure? Reading in a foreign language 8, pp. 689–689.

[57] Kincaid J.P., Fishburne Jr R.P., Rogers R.L. & Chissom B.S. (1975) Derivation of new readability formulas (automated readability index, fog count and flesch reading ease formula) for navy enlisted personnel .

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

[59] Ebert C. & Jones C. (2009) Embedded software: Facts, figures, and future. Computer 42.

[60] Ziv H., Richardson D. & Klösch R. (1997) The uncertainty principle in software engineering. In: submitted to Proceedings of the 19th International Conference on Software Engineering (ICSE'97).

[61] Davis A., Overmyer S., Jordan K., Caruso J., Dandashi F., Dinh A., Kincaid G., Ledeboer G., Reynolds P., Sitaram P. et al. (1993) Identifying and measuring quality in a software requirements specification. In: Software Metrics Symposium, 1993. Proceedings., First International, IEEE, pp. 141–152.

[62] Tjong S.F. (2008) Avoiding ambiguity in requirements specifications. no. February .

[63] Denger C., Berry D.M. & Kamsties E. (2003) Higher quality requirements specifications through natural language patterns. In: Software: Science, Technology and Engineering, 2003. SwSTE'03. Proceedings. IEEE International Conference on, IEEE, pp. 80–90.

[64] Soni M. & Thakur J.S. (2018) A systematic review of automated grammar checking in english language. arXiv preprint arXiv:1804.00540 .

[65] IEEE (1998), IEEE guide for developing system requirements specifications.

[66] IEEE (1998), Ieee recommended practice for software requirements specifications. URL: `http://ieeexplore.ieee.org/xpls/abs\_all.jsp?arnumber=720574`.

[67] Mavin A. & Wilkinson P. (2010) Big ears (the return of" easy approach to requirements engineering"). In: Requirements Engineering Conference (RE), 2010 18th IEEE International, IEEE, pp. 277–282.

[68] Lúcio L., Rahman S., Abid S.B. & Mavin A. Ears-ctrl: Generating controllers for dummies .

[69] Lúcio L., Rahman S., Cheng C.H. & Mavin A. (2017) Just formal enough? automated analysis of ears requirements. In: NASA Formal Methods Symposium, Springer, pp. 427–434.

[70] Tichy W.F., Landhäußer M. & Körner S.J. (2015) nlrpBENCH: a benchmark for natural language requirements processing. Gesellschaft für Informatik eV.

[71] Vadas D. & Curran J.R. (2011) Parsing noun phrases in the penn treebank. Computational Linguistics 37, pp. 753–809.

[72] (2008), Fide handbook – e.i.01a. laws of chess. URL: `http://www.fide.com/component/handbook`.

[73] Braden R. (1989) RFC 1122 Requirements for Internet Hosts - Communication Layers. Internet Engineering Task Force. URL: `http://tools.ietf.org/html/rfc1122`.

[74] Abrial J.R. (1996) Steam-boiler control specification problem, Springer Berlin Heidelberg, Berlin, Heidelberg. pp. 500–509. URL: https://doi.org/10.1007/BFb0027252.

[75] Austen J. (1998) Pride and Prejudice, vol. 1342. URL: ftp://uiarchive.cso.uiuc.edu/pub/etext/gutenberg/etext98/pandp10.zip.

[76] Corpus of Contemporary American English. https://corpus.byu.edu/coca/. Accessed: 2018-11-26.

[77] British National Corpus. https://corpus.byu.edu/bnc/. Accessed: 2018-11-26.

[78] Piantadosi S.T. (2014) Zipf's word frequency law in natural language: A critical review and future directions. Psychonomic bulletin & review 21, pp. 1112–1130.

[79] Lü L., Zhang Z.K. & Zhou T. (2010) Zipf's law leads to heaps' law: Analyzing their relation in finite-size systems. PloS one 5, p. e14139.

[80] Oxford Learner's Dictionaries: The Oxford Text Checker. https://www.oxfordlearnersdictionaries.com/oxford_3000_profiler.html. Accessed: 2018-12-03.

[81] Rathore Y., Ahirwar M.K. & Pandey R. (2013) A brief study of data compression algorithms. International Journal of Computer Science and Information Security 11, p. 86.

[82] Ghosh S., Elenius D., Li W., Lincoln P., Shankar N. & Steiner W. (2014) Automatically extracting requirements specifications from natural language. CoRR, abs/1403.3142 .

[83] Smith R.L., Avrunin G.S., Clarke L.A. & Osterweil L.J. (2002) Propel: an approach supporting property elucidation. In: Proceedings of the 24th International Conference on Software Engineering, ACM, pp. 11–21.

[84] Autili M., Grunske L., Lumpe M., Pelliccione P. & Tang A. (2015) Aligning qualitative, real-time, and probabilistic property specification patterns using a structured english grammar. IEEE Transactions on Software Engineering 41, pp. 620–638.

[85] Bouquet F., Jaffuel E., Legeard B., Peureux F. & Utting M. (2005) Requirements traceability in automated test generation: application to smart card software validation. In: ACM SIGSOFT Software Engineering Notes, vol. 30, ACM, vol. 30, pp. 1–7.

[86] Shadbolt N., Berners-Lee T. & Hall W. (2006) The semantic web revisited. IEEE intelligent systems 21, pp. 96–101.

[87] Bakar N.H., Kasirun Z.M. & Salleh N. (2015) Feature extraction approaches from natural language requirements for reuse in software product lines: A systematic literature review. Journal of Systems and Software 106, pp. 132–149.

[88] Diamantopoulos T., Roth M., Symeonidis A. & Klein E. (2017) Software requirements as an application domain for natural language processing. Language Resources and Evaluation 51, pp. 495–524.

[89] Ilieva M. & Ormandjieva O. (2005) Automatic transition of natural language software requirements specification into formal presentation. In: International Conference on Application of Natural Language to Information Systems, Springer, pp. 392–397.

Appendix A: Sentence analysis

Tokens in sentence

(Sentence length)

	Dataset 1		Dataset 2		Dataset 3		Dataset 4	
	preproc.	converted	preproc.	converted	preproc.	converted	preproc.	converted
Mean	21.5	16.2	25.5	16.8	21.2	14.4	16.4	11
Median	19	14	23	15.5	19.5	12.5	14	9
Mode	18	14	21	11	23	4	7	8
Standard deviation	12.7	9.9	11.7	8.1	9.9	8.7	11.3	6.6

Branches in sentence

	Dataset 1		Dataset 2		Dataset 3		Dataset 4	
	preproc.	converted	preproc.	converted	preproc.	converted	preproc.	converted
Mean	18.2	14.4	19.8	12.8	17.7	12.1	13.5	9.4
Median	16.5	12	18	12	16.5	10	11	8
Mode	11	12	18	8	18	8	7	6
Standard deviation	10	16.1	9.2	5.9	8.7	6.7	9.4	5.2

NPCs in sentence

	Dataset 1		Dataset 2		Dataset 3		Dataset 4	
	preproc.	converted	preproc.	converted	preproc.	converted	preproc.	converted
Mean	5.5	4	5.8	3.8	5.1	3.5	4.5	2.8
Median	5	4	5	4	5	3	4	2
Mode	4	3	5	3	5	2	2	2
Standard deviation	3.3	2.3	2.7	1.9	2.7	2.2	3.1	1.8

VPCs in sentence

	Dataset 1		Dataset 2		Dataset 3		Dataset 4	
	preproc.	converted	preproc.	converted	preproc.	converted	preproc.	converted
Mean	1.9	1.6	1.9	1.5	1.6	1.4	1.6	1.3
Median	2	1	2	1	1	1	1	1
Mode	1	1	1	1	1	1	1	1
Standard deviation	1.2	1.1	0.9	0.7	0.8	0.7	0.9	0.6

Appendix A: Sentence analysis

NPC:VPC in sentence

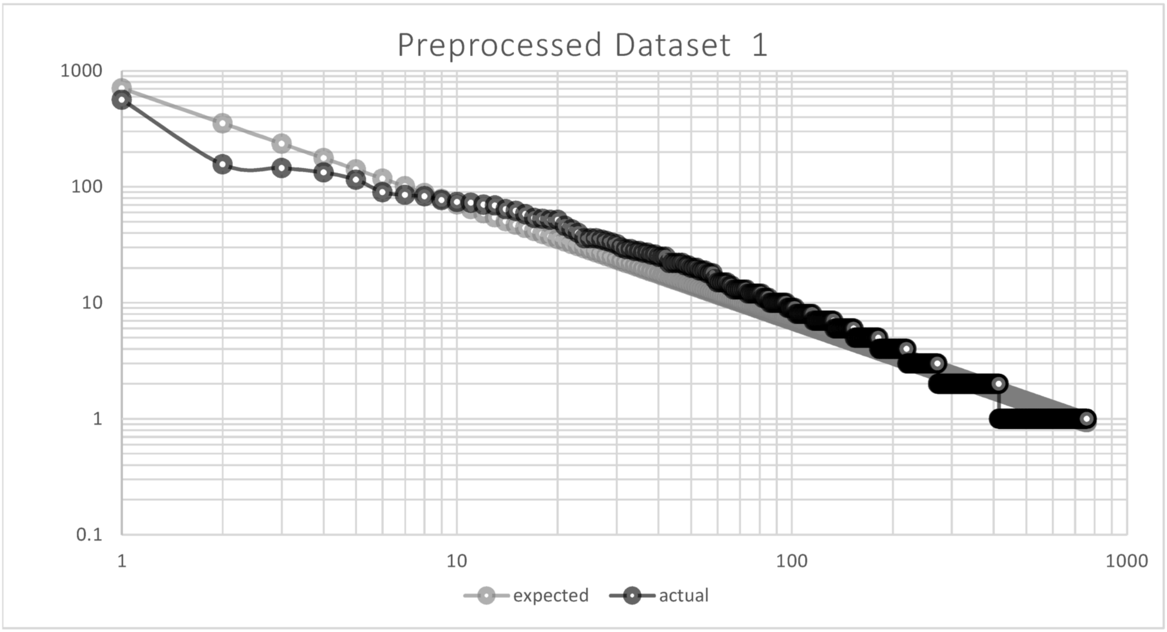
	Dataset 1		Dataset 2		Dataset 3		Dataset 4	
	preproc.	converted	preproc.	converted	preproc.	converted	preproc.	converted
Mean	3.2	2.6	3.4	2.7	3.4	2.5	2.9	2.2
Median	3	2	3	2.5	3	2	2.7	2
Mode	2	2	3	2	2	2	2	2
Standard deviation	1.6	1.3	1.6	1.4	1.9	1.3	1.6	1.2

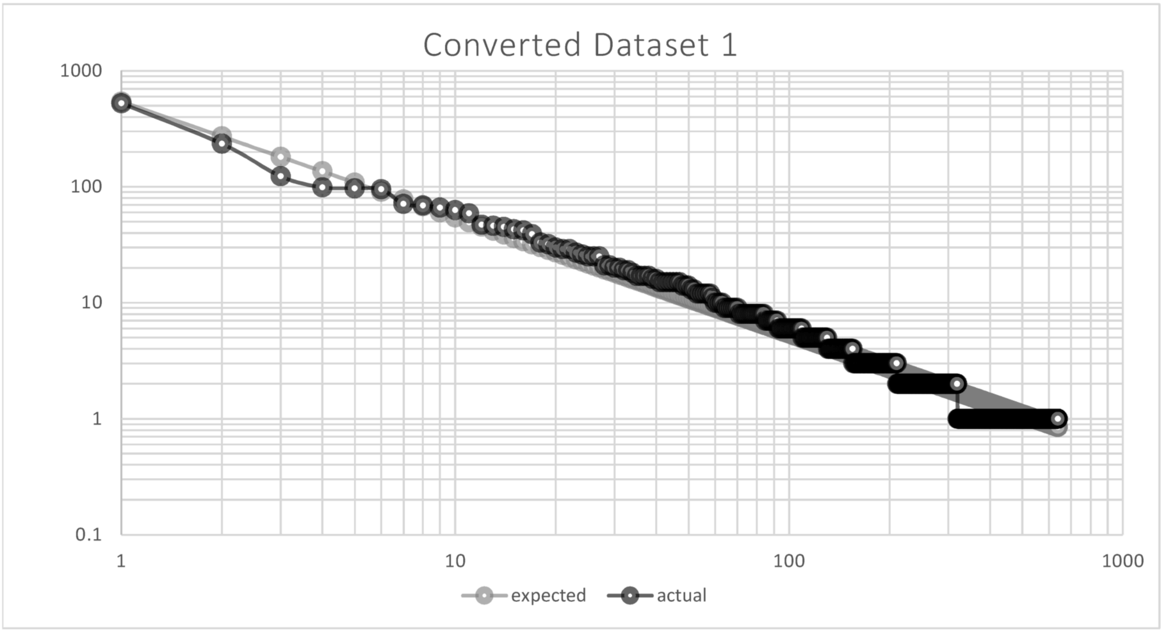
Words in Oxford word lists

	Dataset 1		Dataset 2		Dataset 3		Dataset 4	
	preproc.	converted	preproc.	converted	preproc.	converted	preproc.	converted
Checked words	4980	3779	9454	6454	1756	1227	121413	76828
Top 3000 (%)	94	92	84	81	94	94	90	90
Top 2000 (%)	88	87	75	73	90	90	87	86
Academic (%)	5	5	13	12	12	12	2	2

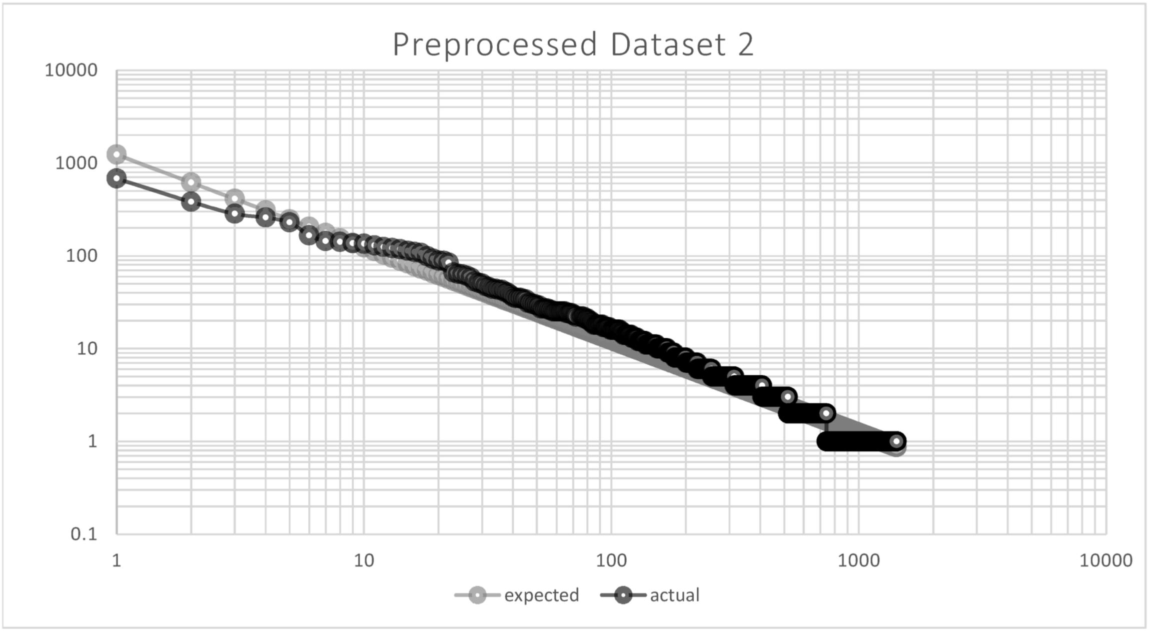
Appendix B: Zipf Plots

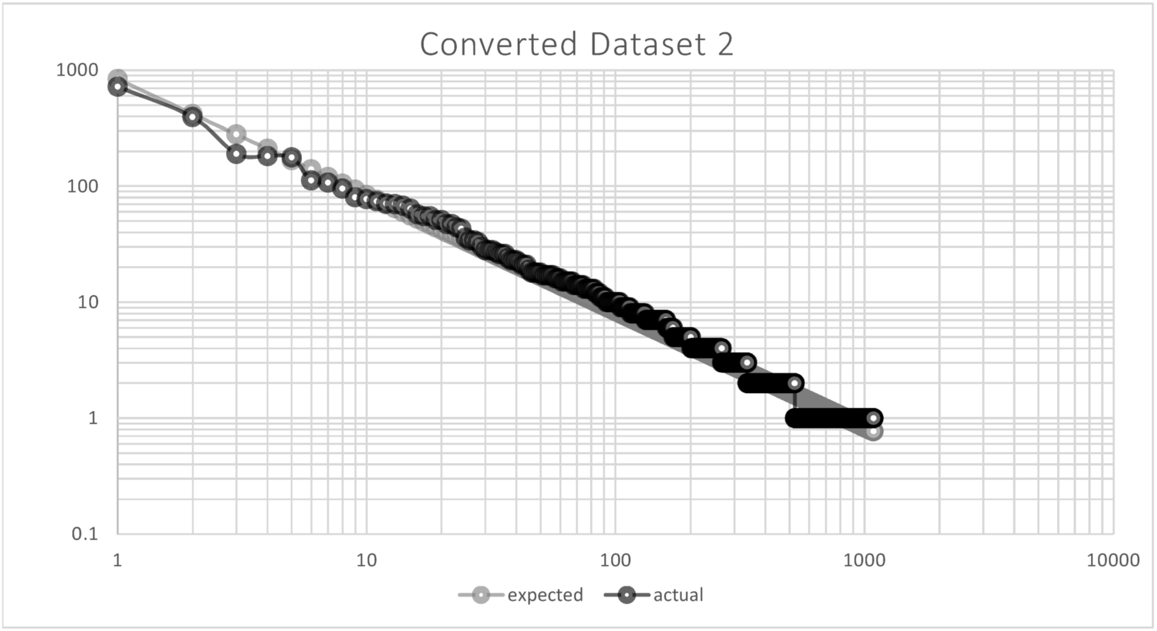
Zipf plots of each dataset before and after conversion.
The ranks of words are on x axis and the counts of the words are on y axis.
Both axes are log scaled.



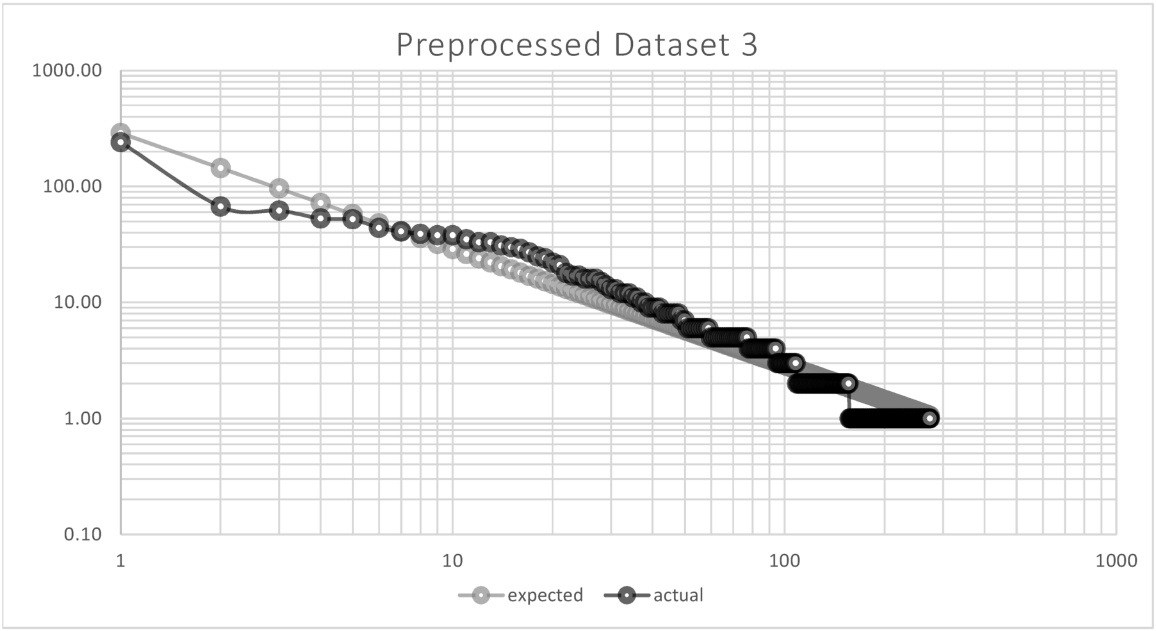


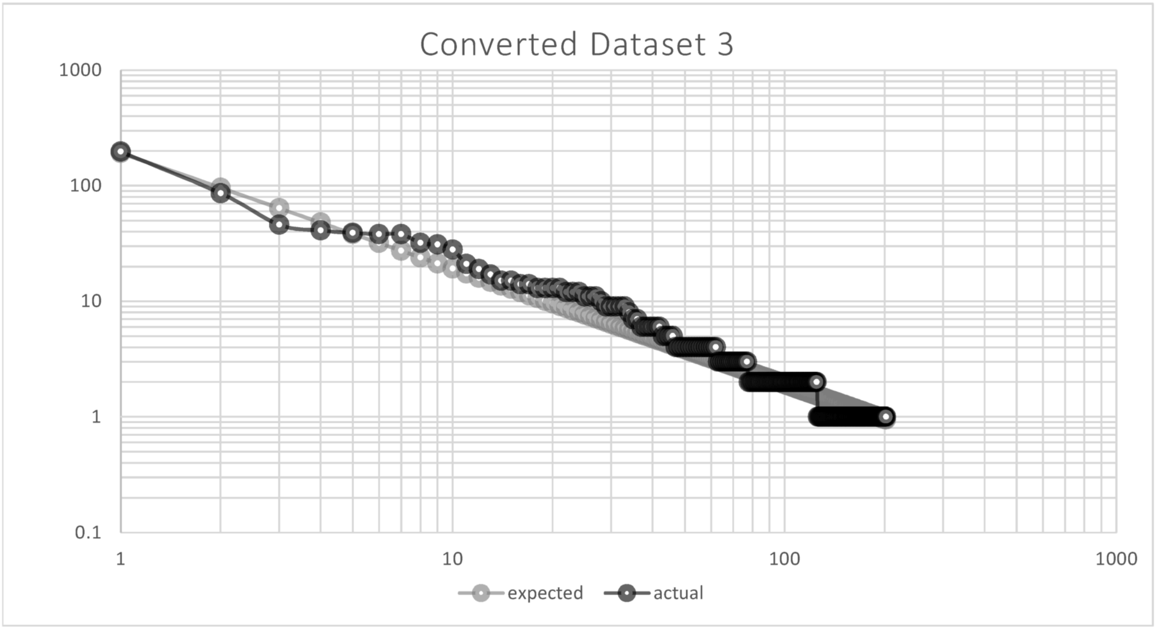
Appendix B: Zipf Plots



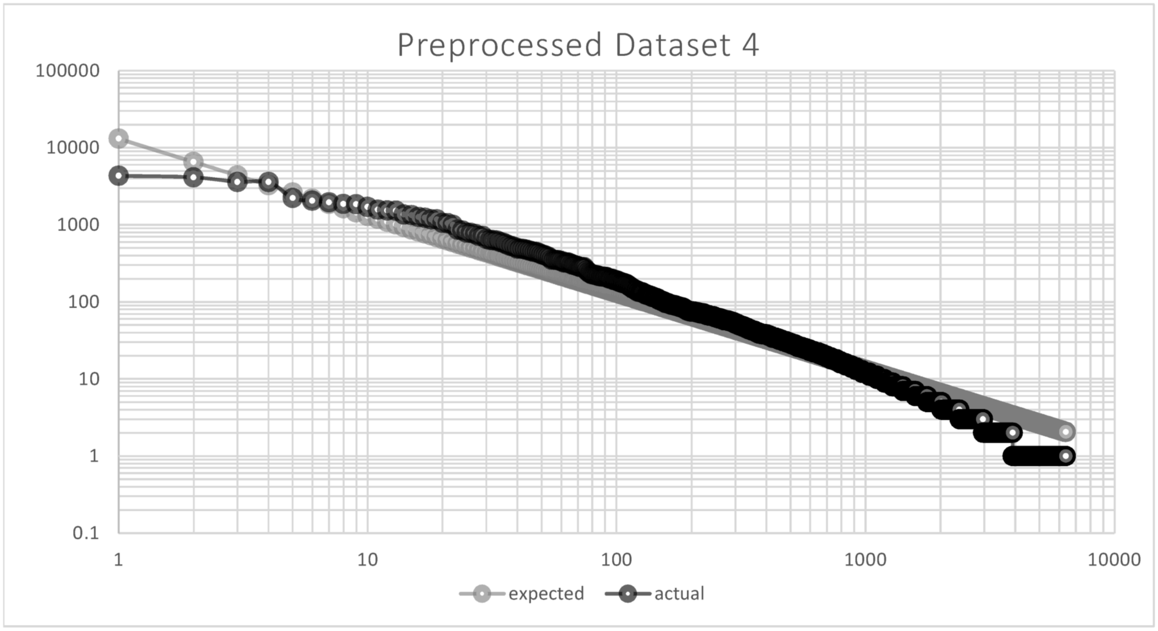


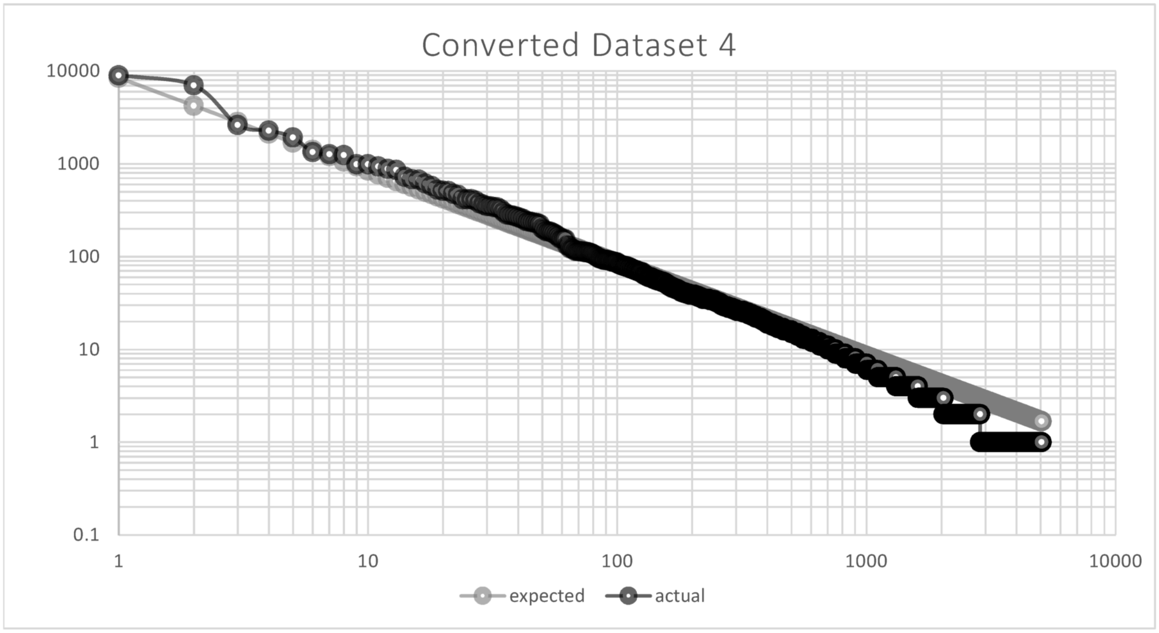
Appendix B: Zipf Plots





Appendix B: Zipf Plots





Appendix C: Most common words

Preprocessed data	Top 20 words	marker words:	**marker**
		stop words:	stop

Dataset 1		Dataset 2		Dataset 3		Dataset 4		
the	564	**the**	679	**the**	241	**the**	4327	1
a	156	a	381	program	67	to	4135	2
of	145	to	284	of	62	of	3607	3
to	133	be	258	is	53	and	3579	4
is	115	must	229	message	52	her	2228	5
player	90	of	166	a	44	i	2063	6
move	85	an	144	physical	41	a	1957	7
game	83	may	141	to	39	in	1866	8
if	77	in	136	units	38	was	1845	9
or	74	is	135	this	38	she	1709	10
shall	73	and	129	mode	35	that	1578	11
in	70	ip	124	has	33	not	1537	12
be	69	that	121	by	33	it	1527	13
his	64	for	118	that	31	you	1359	14
by	62	host	113	which	30	he	1338	15
and	58	should	110	unit	29	his	1268	16
has	54	address	107	water	27	be	1235	17
arbiter	53	not	99	been	25	as	1179	18
on	52	tcp	93	sent	24	had	1176	19
that	52	it	89	failure	22	for	1057	20

Converted data	Top 20 words	marker words:	**marker**
		stop words:	stop

Dataset 1		Dataset 2		Dataset 3		Dataset 4		
the	13.74	**the**	11.33	**the**	16.06	**the**	11.66	1
shall	6.15	**shall**	6.19	**shall**	7.01	**shall**	9.09	2
be	3.22	be	2.99	program	3.75	be	3.39	3
to	2.59	to	2.86	message	3.34	to	2.96	4
a	2.54	a	2.78	be	3.18	of	2.50	5
of	2.49	of	1.76	to	3.10	and	1.73	6
player	1.86	an	1.68	of	3.10	i	1.65	7
then	1.81	ip	1.49	mode	2.61	her	1.61	8
if	1.73	host	1.26	physical	2.53	a	1.28	9
game	1.65	tcp	1.21	units	2.28	she	1.28	10
move	1.54	for	1.16	a	1.71	in	1.22	11
his	1.23	in	1.12	water	1.55	not	1.15	12
in	1.20	address	1.10	sent	1.39	it	1.11	13
or	1.18	and	1.07	stop	1.22	you	0.94	14
arbiter	1.13	layer	1.01	level	1.22	he	0.89	15
by	1.10	**if**	0.90	failure	1.14	his	0.87	16
and	1.02	**when**	0.86	by	1.14	os	0.79	17
on	0.86	**then**	0.86	in	1.06	with	0.75	18
it	0.84	it	0.80	is	1.06	had	0.68	19
not	0.79	not	0.80	**then**	1.06	have	0.66	20
	%		%		%		%	